ESTIMATION AND COORDINATION OF SEQUENCE PATTERNS FOR FREQUENCY HOPPING DYNAMIC SPECTRUM ACCESS NETWORKS

THESIS

Curtis C. Medve, Captain, USAF

AFIT-ENG-14-M-52

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-14-M-52

ESTIMATION AND COORDINATION OF SEQUENCE PATTERNS FOR FREQUENCY HOPPING DYNAMIC SPECTRUM ACCESS NETWORKS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Curtis C. Medve, B.S.E.E.

Captain, USAF

March 2014

AFIT-ENG-14-M-52

ESTIMATION AND COORDINATION OF SEQUENCE PATTERNS FOR FREQUENCY HOPPING DYNAMIC SPECTRUM ACCESS NETWORKS

Curtis C. Medve, B.S.E.E.
Captain, USAF

Approved:

//signed//	6 Mar 2014
LTC Robert J. McTasney, PhD (Chairman)	Date
//signed//	28 Feb 2014
Kenneth M. Hopkinson, PhD (Member)	Date
//signed//	28 Feb 2014
Maj Mark D. Silvius, PhD (Member)	Date

AFIT-ENG-14-M-52

Abstract

In 2010, the Shared Spectrum Company showed in a survey of Radio Frequency (RF) bands that underutilization of spectrum has resulted from current frequency management practices. Traditional frequency allocation allows large bands of licensed spectrum to remain vacant even under current high demands. Cognitive radio's (CR) use of Dynamic Spectrum Access (DSA) enables better spectrum management by allowing usage in times of spectrum inactivity. This research presents the CR problem of rendezvous for fast Frequency Hopping Spread Spectrum (FHSS) networks, and examines protocols for disseminating RF environment information to coordinate spectrum usage. First, Gold's algorithm is investigated as a rendezvous protocol for networks utilizing fast frequency hopping. A hardware implementation of Gold's algorithm on a Virtex-5 Field Programmable Gate Array (FPGA) is constructed to determine the resource requirements and timing limitations for use in a CR. The resulting design proves functionality of the algorithm, and demonstrates a decrease in time-to-rendezvous over current methods. Once a CR network is formed, it must understand the changing environment in order to better utilize the available spectrum. This research addresses the costs a network incurs to coordinate such environment data. Three exchange protocols are introduced and evaluated via simulation to determine the best technique based on network size. The resulting comparison found that smaller networks function best with polled or time-division based protocols where radios always share their environment information. Larger networks, on the other hand, function best when a dispute-based exchange protocol was utilized. These studies together conclude that the selection of a rendezvous algorithm or a protocol for the exchange of environment data in a CR network are determined by the characteristics of the network, and therefore their selection requires a cognitive decision.

I dedicate to my Mother and Sister. Their never ending support helps guide me to push all aspects of my life toward the better.

Table of Contents

Page

Page

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
AFIT	Air Force Institute of Technology
CCC	Common Control Channel
COD	Code-of-the-Day
CR	Cognitive Radio
DCF	Distributed Coordination Function
DSA	Dynamic Spectrum Access
FCC	Federal Communications Commission
FFH	Fast Frequency Hopping
FFT	Fast Fourier Transform
FHSS	Frequency Hopping Spread Spectrum
FPGA	Field-Programmable Gate Array
FPSFH	*full-packet* slow frequency hopping
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
LFSR	Linear Feedback Shift Register
LUT	Look-up table
MAC	Medium Access Control
MTTR	maximum TTR
NTIA	National Telecommunications and Information Administration
OSI	Open Systems Interconnection
PCF	Point Coordination Function
PRNG	Pseudo-Random Number Generator

Acronym	Definition
REM	Radio Environment Map
RF	Radio Frequency
SFH	Slow Frequency Hopping
SMREM	Suggested Master REM
SPSFH	*sub-packet* slow frequency hopping
TTR	Time-to-Rendezvous
VHDL	Very High Speed Integrated Circuit Hardware Description Language

ESTIMATION AND COORDINATION OF SEQUENCE PATTERNS FOR FREQUENCY HOPPING DYNAMIC SPECTRUM ACCESS NETWORKS

I. Introduction

Continuing development of wireless systems causes increasing stress on the available spectrum resources in the United States and other countries throughout the world. The National Telecommunications and Information Administration (NTIA) and the Federal Communications Commission (FCC) control the spectrum availability and usage in the United States. NTIA controls frequencies assigned to federal agencies while the FCC administers commercial and state use. In November 2010 the FCC released a notice of inquiry focusing on dynamic spectrum access technologies to enable more efficient utilization of the limited spectrum. The notice presented several federal and non-federal programs that investigate possible solutions to alleviate the spectrum. One such technology is referred to as Cognitive Radio (CR) [10].

The NTIA in [25] defines a CR system as "a radiocommunication system that is aware of its environment and internal state and can make decisions about, and adjust, its operating characteristics based on information and predefined objectives."[25] CR is the enabling technology for Dynamic Spectrum Access (DSA) which is the operation of changing spectrum usage based upon the state of the Radio Frequency (RF) environment. DSA is currently a topic of focus for researching bodies around the world to include the Air Force Institute of Technology (AFIT). Research at AFIT investigates the use of DSA for both civilian and military applications while advancing technology in the area of radio communication.

1.1 Motivation

The Air Force relies heavily on wireless communication systems. Communication between command centers, personnel, and other military platforms is vital for successful operations. Twelve core functions comprise the US Air Force: Nuclear Deterrence Operations, Special Operations, Air Superiority, Global Integrated ISR, Space Superiority, Command and Control, Cyberspace Superiority, Personnel Recovery, Global Precision Attack, Building Partnerships, Rapid Global Mobility and Agile Combat Support [31]. Each of these functions greatly depend on continuous and reliable communication. As such, the Air Force has outlined the current state of capabilities, assessed projected challenges, and presented a plan to address problems. This Air Force report discusses "Frequency Agile Spectrum Utilization", a sub-topic of DSA, as a potential capability area [3].

Military radio requirements vary widely from commercial requirements due the range of operational environments and higher reliability requirements. For instance, Frequency Hopping Spread Spectrum (FHSS), used in many military systems, is a method of transmission not commonly used by commercial radio systems due to the lower data rates they provide. This addition of FHSS to the problem space brings separate challenges to the study of CR for military use.

1.2 Problem Statement

The objective of this research is to investigate Gold's algorithm as a rendezvous method for FHSS radio systems utilizing DSA, and to evaluate the network performance losses caused by exchanging radio environment data. Explored separately, research into these topics areas expand upon previous AFIT cognitive radio research designed to construct and evaluate a prototype CR.

For any network to begin functioning, the network itself must be established. Rendezvous is the process of establishing a communication link between two or more

radios or radio networks. Current research on the rendezvous problem ignores the functionality of FHSS within a DSA environment. The research in this document addresses this missing consideration. It also investigates the hardware specifications and timing limitations for implementing a unique rendezvous algorithm for FHSS on an Field-Programmable Gate Array (FPGA) based development platform.

In addition to establishing a network connection, radios must coordinate a common picture of the environment. The exchange of environment data requires additional data transfers as overhead which reduces the overall performance of a network. Network performance describes the transfer characteristics of non-control data to include throughput, delay, and data drop rates. The exchange of environment data occurs when needed or on a set schedule. This research investigates the overhead cost of performing an exchange under different conditions. A lack of research into exchange protocol effects suggest that no standard metrics exist to identify the efficiency of a protocol, nor do any test data sets exist to represent real-world environments for testing of a protocol.

The specific goals of this research are to:

- demonstrate that Gold's algorithm allows for successful rendezvous on a FHSS radio network without dwell time or common control channel requirements.
- present resource requirements and timing limitations for an FPGA implementation of Gold's algorithm.
- analytically show that the above algorithm is capable of operating within the DSA paradigm.
- evaluate the performance of three possible exchange protocols for transferring radio environment data on a multi-nodal radio network via wireless network simulation.

1.3 Research Contributions

This research provides two distinct contributions to the study of DSA, and therefore the field of CR. First, it establishes Gold's algorithm as a valid rendezvous method for Frequency Hopping DSA networks. Compared to traditional rendezvous methods such as Code-of-the-Day (COD), Gold's algorithm reduces joining time when higher hop rates are required. Additionally, this research offers the first FPGA hardware implementation of Gold's algorithm providing timing and resource requirements for radio designers.

Second, this research demonstrates the impact to network performance for three exchange protocols when radio channel availability data must be exchanged as part of DSA operations. Examination of metrics such as data throughput, delay, and data drop rates provide an estimation of network performance with varying network size under different network non-control traffic workloads. A comparison of these metrics and their correlations determine the exchange protocol that minimizes the overhead costs to the network.

Two OPNET nodal models were developed to compare the exchange protocols. Utilization of these models provide future researchers with advanced capabilities not currently available. The first model expands the features of the Institute of Electrical and Electronics Engineers (IEEE) 802.11b OPNET wireless model to incorporate the transmission technique of FHSS. OPNET's model provides frequency hopping as a function option, but the feature was never implemented in the model. The improved model is currently restricted to slow-hopping operations. The second model modifies the Point Coordination Function (PCF) of IEEE 802.11b within the Medium Access Control (MAC) process model. This modification allows PCF operation without the need of a dedicated access point, and leverages the PCF operation for transfer of radio environment data only.

1.4 Thesis Organization and Overview

Chapter 2 examines related work on topics within cognitive radio, and provides background information. It begins with an overview of cognitive radio and spectrum allocation terminology. The chapter continues by providing an overview of frequency hopping spread spectrum, then explores the problems of rendezvous and environment mapping.

Chapter 3 presents the methodology for the research experiments on both Gold's algorithm and the network overhead cost simulations. First, it discusses the development and experimentation of the custom Intellectual Property (IP) core implementation of Gold's algorithm to determine system physical and timing specifications. Secondly, the chapter describes three potential exchange protocols for transferring environmental data, and the methodology for testing these protocols for network performance using wireless network simulation. Finally, the analysis approach and hypothesis of the experiments is disclosed.

Chapter 4 presents and analyzes the results for the implementation of Gold's algorithm and network simulation experiments. The analysis for Gold's algorithm considers timing limitations, resource requirements, and expected rendezvous performance. Additionally, a discussion on translating hop sequences to usable channels in a DSA environment is presented. Thereafter, the chapter examines the network simulation results, and presents their impact on performance.

Chapter 5 summarizes the conclusions of this research, discussing its contributions, and providing suggestions for future work. Appendices containing experimental results, a detailed explanation of Gold's algorithm, an example of Gold's algorithm, and Very High Speed Integrated Circuit Hardware Description Language (VHDL) code are attached at the end of the document.

II. Related Work

Dr. Joseph Mitola first introduced cognitive radio in 1998 as part of his doctoral dissertation [23][24]. This dissertation defined the early concepts of cognitive radio in terms of feature requirements and a possible architecture. As part of early development, Mitola attempted to classify characteristics of radio cognition into the nine levels shown in Table 2.1.

Table 2.1: Mitola's Characteristics of Radio Cognition Tasks [24]

Level	**Capability**	**Task Characteristics**
0	Pre-programmed	The radio has no model-based reasoning capability
1	Goal-driven	Goal-driven choice of RF band, air interface, and protocol
2	Context Awareness	Infers external communications context (minimum user involvement)
3	Radio Aware	Flexible reasoning about internal and network architectures
4	Capable of Planning	Reasons over goals as a function of time, space, and context
5	Conducts Negotiations	Expresses arguments for plans/ alternatives to user, peers, networks
6	Learns Fluents	Autonomously determines the structure of the environment
7	Adapts Plans	Autonomously modifies plans as learned fluents change
8	Adapts Protocols	Autonomously proposes and negotiates new protocols

Concept development using these levels progressed through the exceptional efforts of researchers around the world. Vanu Bose with an alternate view of cognition described his vision of a fully programmable radio to improve spectrum usage as "dynamically adapting the physical layer of the network to best meet the current environmental conditions, network traffic constraints and application requirements, rather than a lowest common denominator service that must accommodate the worst case."[6] No matter the vision of cognitive radio, common themes arose in areas of interest. The research presented in this paper focuses on the concept of DSA to utilize the spectrum more efficiently. Level 6 best

represents DSA in Mitola's characteristics. Putting DSA and CR into use first requires an understanding of current spectrum management practices.

The FCC manages and regulates all domestic non-federal spectrum under Title 47 US Code 301. The FCC currently divides the spectrum into two categories, Unlicensed Spectrum and Licensed Spectrum for Commercial Services. Unlicensed Spectrum are frequencies designated as "unlicensed" or "licensed-exempt" where any user can operate devices without the need of an FCC license, but must use certified radio equipment that complies with FCC requirements. Since no license is required, operation in this spectrum typically involves dealing with levels of interference. Frequencies such as in the Industrial, Scientific and Medical (ISM) radio bands and the Unlicensed National Information Infrastructure (U-NII) radio band fall under the unlicensed category [11]. Licensed Spectrum designated for commercial services allow exclusive ownership or use of particular frequencies. Locality of transmission also affects the restricted use of licensed spectrum.

As the controlling authority, FCC reports on a constant basis the allocations of spectrum to different users. Figure 2.1 show a summary break down of the 300-3000 MHz frequency span. The assignment of spectrum using this two category system does not ensure optimal use of that spectrum; therefore, to make any improvements the current usage must be understood.

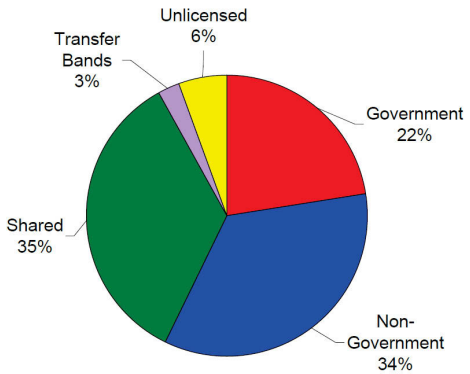


Figure 2.1: Division of Spectrum Between Government and Non-Government [38]

Starting in 2004, the Shared Spectrum Company has collected spectrum data in several US cities to investigate current usage [20, 21, 32]. These collection studies showed many of the channels reserved as licensed spectrum are underutilized. Figure 2.2 shows the reported the usage over time of 2.4 GHz to 2.5 GHz in the study of Vienna, VA. Under utilization of the spectrum is a large catalyst for the development of DSA.

McLean et al. attempted to highlight the functions needed in implement a working DSA radio system [22]. Specifically, they present a list of functions required for the cognitive process to maintain communication. Figure 2.3 graphically represents these functions.

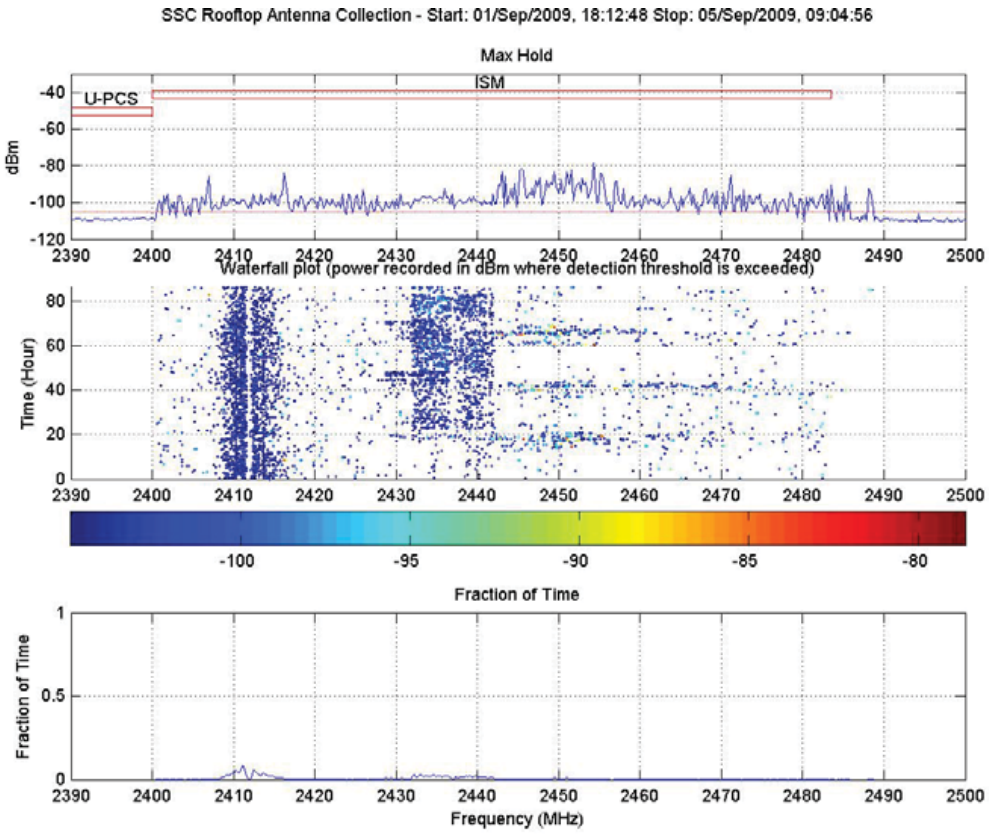


Figure 2.2: 2.4GHz to 2.5GHz usage Vienna, VA [32]

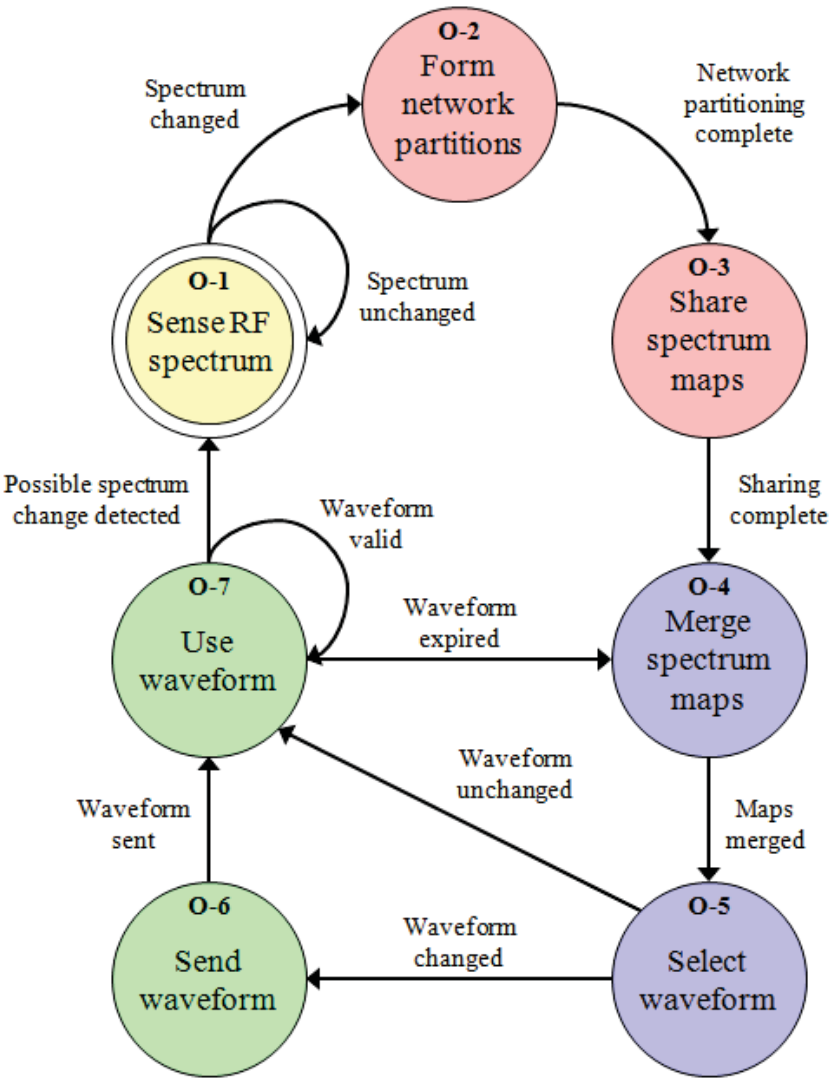


Figure 2.3: Cognitive radio system function diagram [22]

2.1 Frequency Hopping

The term *frequency hopping* exists throughout cognitive radio research. This section attempts to categorize the different types of frequency hopping in order to differentiate the DSA scenarios.

Traditional frequency hopping describes any radio mode of operation where the communication channel changes over time. For example, a radio may operate on Channel #1, and increment its channel by one channel every minute. The act of automatically changing frequencies according to a sequence or algorithm defines frequency hopping; however, distinguishing the amount of information that can be exchanged while occupying a single frequency divides frequency hopping into categories. Radio communications developers use two common categories, Fast Frequency Hopping (FFH) and Slow Frequency Hopping (SFH) [26].

In FFH, the hop rate (rate of frequency change) exceeds the symbol rate causing a single data symbol to be cast over multiple frequency channels during transmission. Although traditionally resulting in lower data transfer rates, FFH provides a level of bit error correction based on the ability to compare a data symbol over the multiple frequencies. For instance, if a data symbol is transmitted over five frequencies then the received majority is likely to represents the true value that was transmitted.

A system is considered to be SFH if the hop rate is less than the data symbol rate. Most commercial radios use this type of frequency hopping for use in IEEE 802.11 for Wireless Local Area Networks and IEEE 802.15 for Bluetooth technologies. The definition of SFH contains an ambiguity when classifying a radio transmission by the quantity of data transferred. The definition covers a radio capable of transmitting as little as a single data symbol or more than an entire packet before changing frequencies. As such, a new division of SFH into more specific categories is needed. Distinguishing the ability to establish reliable communication channel, typically a resulting from a

"handshake" protocol, produces a possible division point. This document provides terminology to identify this division.

Let *full-packet* slow frequency hopping (FPSFH) define the case when all needed traffic to establish a connection can be transferred during a single time slot, and let *sub-packet* slow frequency hopping (SPSFH) define the case when the exchange can not occur during a single time slot. The definitions of FFH, FPSFH, and SPSFH establish an additional factor for determining the applicability of different DSA routing and rendezvous protocols.

2.2 Problem of Rendezvous

Users within a CR network detect the presence of other users to establish communications links thereby allowing for information exchanges to occur. The process of two or more radios or radio networks establishing a communication link defines rendezvous. Rendezvous requires that the users occupy the same channel at the same time to establish a link. This fundamental task becomes increasingly more difficult as the complexity of the communication medium and communication protocols change. For instance, if a radio user (Radio A) can only operate on a single channel, then any other radio user (Radio B) can easily perform a rendezvous by using the same channel.

As Radio A increases the number of available channels it operates on, Radio B's task of finding the correct channel increases too. Let Radio A select and remain static on one of its N available channels. Radio B must, at most, search each of the N channels before rendezvous occurs. Next, let Radio A change its operating channel according to a set algorithm or sequence. Radio B must determine a way to rendezvous with increased difficulty. This problem becomes non-trivial. Further adding cognitive abilities to a radio brings the dynamic element of a changing number of available frequencies for each user.

This is the essence of the rendezvous problem for CR networks. For evaluation, metrics provide the ability to compare different rendezvous algorithms against one

another. One important metric is Time-to-Rendezvous (TTR), which is defined as the number of time slots required for a rendezvous to occur. However, since the TTR depends on the specific channel order that the radios follows, the maximum TTR (MTTR) metric is commonly used. MTTR is the TTR for the worst case scenario when applying a rendezvous algorithm. The problem space for rendezvous within CR networks is vast, and many algorithms have already been created to address differing applications.

2.2.1 Previous Algorithms.

Rendezvous algorithms can be categorized by the assumptions required for operation, and by the extent which the algorithm's effectiveness can be applied. Liu et al. suggested a taxonomy for subdividing channel hopping rendezvous algorithms based on the target environment [18]. Figure 2.4 displays a visual representation of this taxonomy.

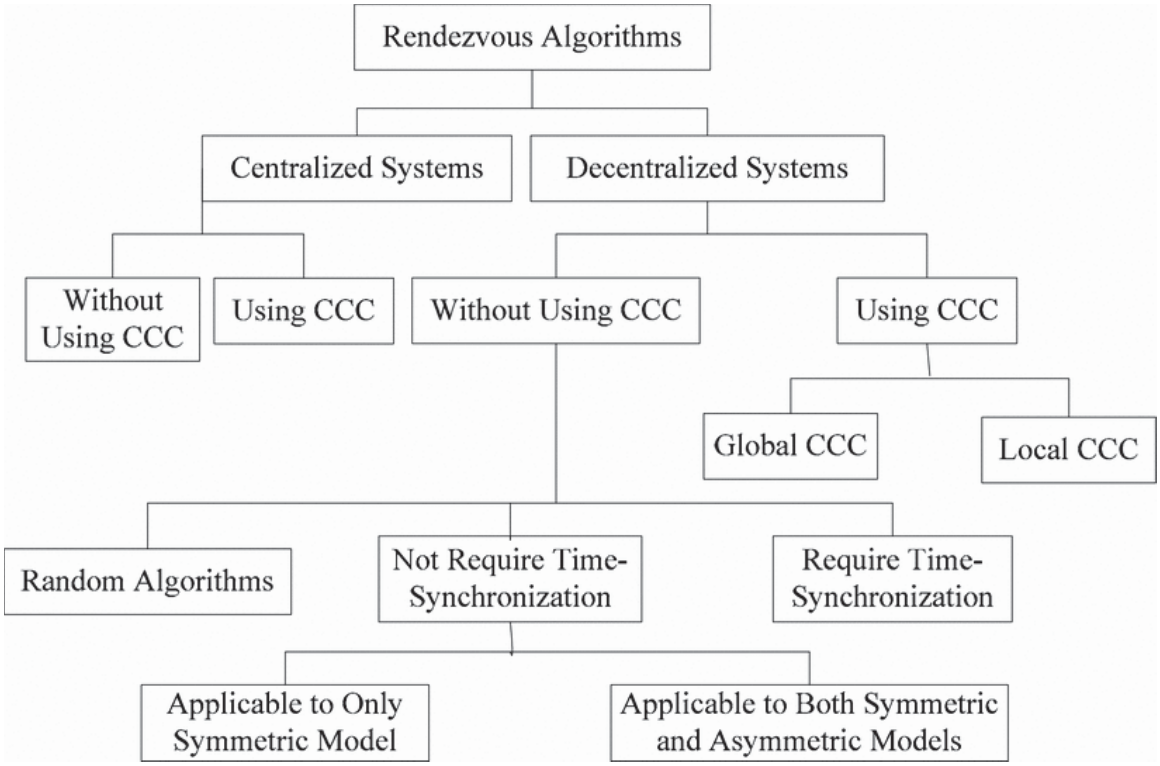


Figure 2.4: Taxonomy of rendezvous algorithms [18]

The top level of the categorization splits by identifying algorithms based on the topography of the system. These two systems are centralized and decentralized. In a centralized system, a user requires a centralized controller (server) to assist users in the rendezvous process. A decentralized system does not require server assistance to rendezvous. Both types of systems may utilize a Common Control Channel (CCC). The inherent limitations of a centralized system led research to focus on decentralized systems. The failure of a server resulting in the crippling the entire network is one such limitation. This research focuses on decentralized systems.

Using a dedicated CCC is the simplest way to coordinate rendezvous. The CCC approach divides time into two intervals: control interval and data interval. During the control interval, users coordinate and select an available channel for data transfer. Figure 2.5 illustrates such an exchange using a CCC. The CCC is assumed to be reachable by every CR (global) or to a select group of CRs (local), normally determined by a clustering algorithm [42].

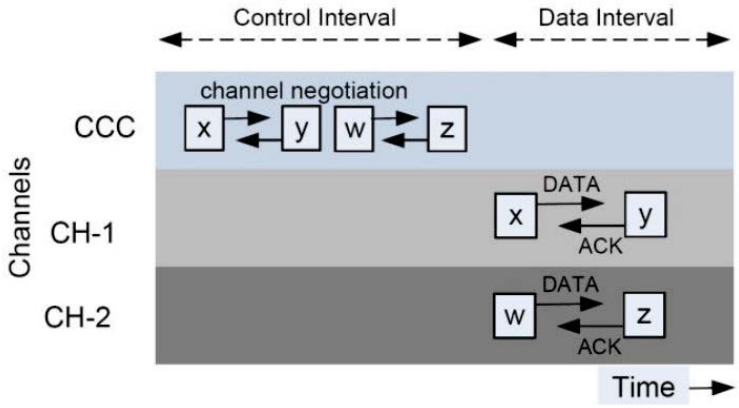


Figure 2.5: Common Control Channel coordination [15]

The use of a CCC causes inherent problems such as the coordination of the control channel itself, the availability of the channel, and its susceptibility to malicious interference. The lack of a CCC is typically addressed as a more complicated problem

known as *blind rendezvous*. When no CCC is needed, algorithms are further divided by the need for time-synchronization on the systems to communicate. Lastly, a division occurs between the environment models in which the algorithm applies. A symmetric model describes the situation where each CR has the same available channels. The asymmetric model represents all other cases.

Research toward the development of CR introduced a number of algorithms. In an attempt to remain robust, research focused on decentralized systems for rendezvous algorithms. A successful channel-hopping (CH) rendezvous protocol meets three basic requirements: [5, 42]

1. Every pair of nodes should have a chance to meet every other node periodically within a bounded interval.

2. All nodes should be able to rendezvous on every channel it is capable of using for communication.

3. All channels should have the same probability to be utilized as the control channel.

More complicated scenarios were generated as algorithm development progressed. A scenario driven view of many of the known rendezvous algorithms provides better understanding of the algorithms and their functions. To establish different scenarios, a set of global assumptions must be made about the network. First, any radio node being considered as part of the system must share at least one channel with the other radios in the system. This means the operational ranges of the radio in the network must overlap. Second, at least one channel must be open within the overlapping channels for communication exchange to occur. Third, radios must be capable of exchanging information once co-existence on a channel occurs.

The simplest scenario within rendezvous is communication between only two radios operating under the symmetric environment model (i.e. all nodes have the same open

channels). The Random Algorithm (RA), the first algorithm investigated, operated by selecting at random an open channel from a list. Random algorithms cannot guarantee the rendezvous of nodes in a bounded time; however, RA is considered the baseline from which other algorithms developed.

Sequence based algorithm design is one method to ensure rendezvous. In [7], DaSilva & Guerreiro proposed Sequence-based Rendezvous (SeqR) capable of linking two radios with an MTTR of $M(M+1)$ time slots. The algorithm does not require time-synchronization, but is limited to the symmetric model.

A second scenario exists by expanding the two node scenario to include the asymmetric model conditions. Sequence rotation algorithms such as Modular Clock and Modified Modular Clock (MMC) displayed the ability to work for both the asymmetric and symmetric models [35]. These algorithms are based on a prime modulus function and a randomly selected hop distance. The algorithms showed a bounded MTTR as long as both the rate and prime number are not selected as the same value for more than one node. A separate effort demonstrated applicability of these algorithms via hardware implementation on the GNU radio test bed [30]. It also presented a comparison of MMC, RA, and a modified MMC.

Expanding off the modular clock design, [17] proposed two ring-walk algorithms. This algorithm added a separate time component where each node may or may not perform a frequency change. The ability to hold on a channel eliminated the limitation that the modular clock presented. As such, the ring-walk algorithm guaranteed rendezvous of two users in a time slot without time-synchronization on both symmetric and asymmetric models. More than two radios may visit the same operating channel in the above scenarios, but the design of the algorithms did not ensure this event to occur.

Bian et al. in [5] demonstrated Quorum-based channel hopping (QCH). Novel to this algorithm class was the insurance that rendezvous occurred on multiple channels within a

single sequence period. This meant that several chances to rendezvous occur. The paper presented a comparison between QCH, RA, and SeqR to demonstrate the multiple overlapping channels effect. Using the QCH approach, Bain et. al introduced three distinct variations: M-QCH to minimize MTTR, L-QCH to minimizes the load on the channels, and A-QCH to handle asynchronous operations.

The usefulness of only two radios communicating quickly loses its practicality when developing complex scenarios. This caused researchers to expand into two different multi-node scenarios, organized hop and multiple pairing. Lin et al. proposed the jump stay (JS) algorithm which functions by dividing the hop sequence into two parts, an active jump pattern and a passive stay pattern. The selection of jump moves are based on a random step distance and starting point (through a modulus function). During stay period a single channel is held. The sequence created is similar to the ring-walk algorithm; however, the JS algorithm extended to multi-user scenarios where more than two nodes can meet on the same channel. As such, [16] presented the organized hop method by which many users can force synchronization of patterns when nodes meet. Two nodes will adopt the same sequence when they rendezvous and hop as an organized unit. As more rendezvous occur the organized pack grows. Since the JS algorithm is bounded it ensured that all nodes eventually exist on the same pattern [16]. This method expands to almost all previous algorithms. In a comparison of known algorithms, JS had the best overall performance among single pair rendezvous algorithms covering both symmetric and asymmetric models according to [18].

Zhang et. al in [42] focused on creating sequences allowing multiple pairings across the spectrum to occur every time slot. In other words, node pairs rendezvous on many of the available channels. The algorithm proposed requires pre-knowledge of the number of nodes in the network and a homogeneous system view. This method also utilized by Xin et. al in creating the Rendezvous with near-Optimal Performance (ROP) algorithm was

designed to distribute the rendezvous load over all available channels [41]. The benefit of this multiple pairing method lays in the number of simultaneous connections which cause the network throughput to greatly increase.

Wu & Wu found little attention had been shown to recognize the difficulties of utilizing radios capable of measuring different spectrum spans [39]. As such, the last significant scenario covers a specific situation using heterogeneous systems. A heterogeneous system refers to radios in the system using different operating ranges, and attempts to communicate. For instance, if radio A operates on frequencies 1 - 2 GHz and needs to communicate with radio B, which operates at 1.5 - 2.5 GHz, then radio A and B overlap operational frequencies in the 1.5 - 2 GHz range. This overlap allows rendezvous to occur. In 2012, research papers began to focus on this new consideration for rendezvous. Wu & Wu addressed this by presenting the interlocking channel hopping (ICH) algorithm which utilizes the intersection of two frequency spans.

2.2.2 Limitations in rendezvous.

Most of the scenarios above allow communication to occur for only a fraction of the operating time. Pre-determined frequency sequences for rendezvous do not allow for constant communication. With the exception of the organized hop method, multiple radios do not maintain an established connection for more than one time slot.

Furthermore, the researched algorithms described above presented common assumptions by the authors. Most algorithms defined a rendezvous as successful when "two nodes access a channel during a certain period of time which is long enough to establish a reliable link."[33]. [2], [4], [5], [35], and [39] expressed this in alternate wording. This imposed the restriction of operating only within FPSFH.

As a focus point, this research investigates rendezvous for FHSS systems which may operate at frequency rates defined by SPSFH and FFH. SPSFH is critical for the

application of military radio operation. As such, this effort explores a robust method of rendezvous capable of supporting both slow and fast frequency hopping.

2.2.3 Sequence Estimation with Gold's time of arrival algorithm.

Only two possibilities exist for rendezvous without the ability to establish a reliable communication channel on a single frequency hop as with FPSFH. Either the the radio must have complete foreknowledge of the network's hopping sequence, or it must have the ability to identify the current hopping sequence of the network. The first case describes traditional radio operation using the COD method. For the latter case, Gold's algorithm provides a process to calculate the hopping sequence of a network by monitoring a single communication channel.

Dr. Robert Gold created the algorithm as part of a Small Business Innovative Research contract with the US Air Force Sensors Directorate, Air Force Research Labs, Wright-Patterson AFB, OH [28][29]. Additionally, Robert Gold Comm Systems Inc patented the algorithm [13]. No academic publications of this algorithm are known.

Golds algorithm functions by estimating the state of a Linear Feedback Shift Register (LFSR) based Pseudo-Random Number Generator (PRNG) sequence. LFSRs are commonly used to create the hop sequences of FHSS systems. The algorithm is designed to measure the time differences between arrivals of a radio system transmission on a single channel. The operation of the algorithm requires specific parameters of the target LFSR system to be known. The parameters include the hopping rate of the target system, the bit length of the LFSR, and which LFSR bits are used as feedback. Requiring knowledge of the parameters limits cross design development; however, current radio systems that use LFSR based PRNG sequences already operate with similar restrictions.

By monitoring a single channel for the arrival times, the algorithm calculates the state of the LFSR bits for which all future states depend. This allows the algorithm to serve as a rendezvous method for establishing communications with an existing SPSFH or

FFH network. A full packet exchange is not required for the algorithm to function. Appendix B provides a full description and example of Gold's algorithm functionality.

Due to the applicability of Gold's algorithm toward current military radio systems, this research establishes the algorithm as an implementable method for FHSS rendezvous across all sub-categories. Additionally, the development of an FPGA implementation of Gold's algorithm within this research enables system hardware requirements to be determined as well as support the creation of an FHSS DSA radio test platform for use by the AFIT's Cognitive Radio Laboratory.

2.3 Environment Map Exchange Overhead

For a DSA enabled radio to perform as part of a network, the radios must coordinate a common picture of the environment in which they operate. The concept of the Radio Environment Map (REM) was introduced to coordinate this environmental data. Originally, an integrated database consisting of layers of radio environment domains formed the REM concept. These domains included geographical features, spectral regulations, radio location, activities of the radio, user policies, and past experiences [43]. Dedicated network sensors, a compilation of spectrum authority databases, and by the very radios attempting to utilize the data funnel information to the REM. Figure 2.6 shows a representation of data fusion to create a REM database. A separate or more specific REM can exist on multiple levels. The REM concept contains a vast amount of information to describe the environment; therefore, the cognitive radio community should research both the methods of acquiring information for the REM and the distribution of that data.

Determining a sensor's local RF environment is arguably the most impacting aspect of developing a REM. REM creation requires transmission detection of other systems in the vicinity of operation. Wellens and Mähönen presented an array of both practical and theoretical spectrum sensing methods [37]. Additionally, [36] provided a comprehensive

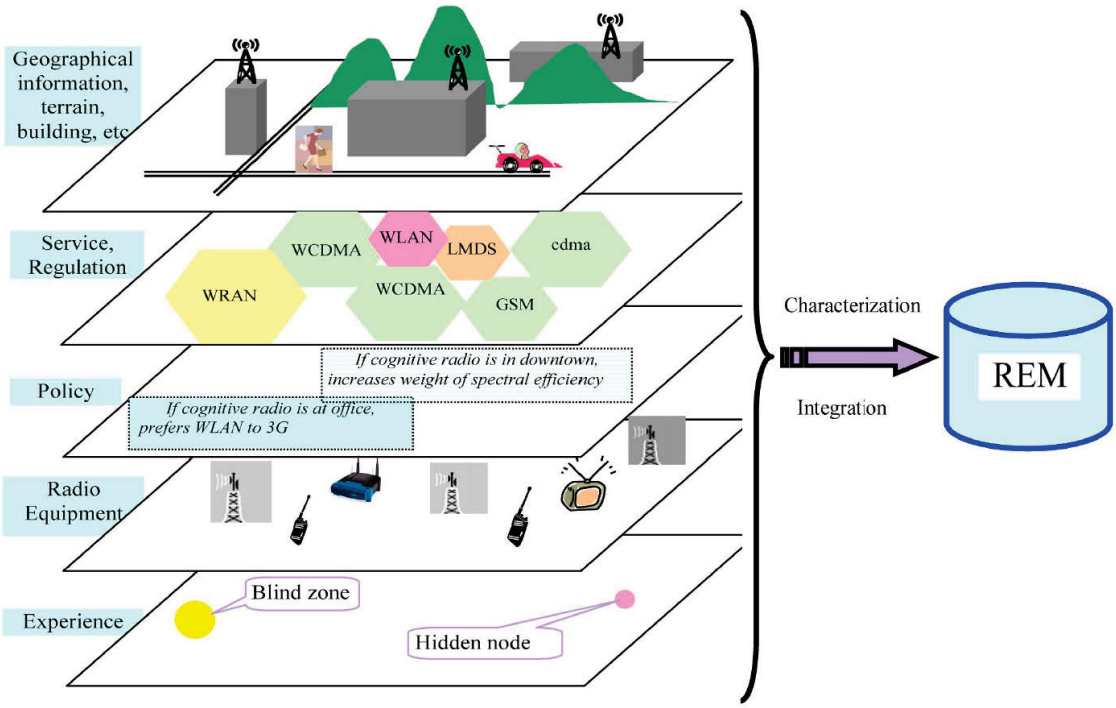


Figure 2.6: REM as an integrated database [44]

summary of current methods of spectrum sensing. Shokri-Ghadikolaei and Fallahi sought to determine an optimal sensing sequence to produce the greatest data throughput [34]. In coordination with the sensing methods, the number of sensors control the accuracy of representing the environment.

Faint et al. calculated the critical number of nodes are needed to create a REM that properly represents an area. The value depended on the total area under investigation and the locality of the sensor nodes [9]. Similarly, Hanif et al. investigated the performance of CR networks when discrepancies exist between the REM being used for operation and the true environment [14].

After *local* REM collection, the network must distribute the REM to the other nodes in the area (decentralized system), or to a *master* node or database (centralized system). Routing protocol research which utilizes REM information is found throughout recent publications. [8] and [19] summarized over 30 different routing protocols showing a major

research focus on the routing of network information to achieve the best performance. However, these routing protocols assumed that the neighboring network nodes already share REM information. Little research addresses the physical dissemination of the REM once collected by the sensor systems. Zhoa et al. in [44] compares REM dissemination schemes on an ad-hoc network. The comparison addressed three schemes: a network flooding scheme, an optimized link state routing protocol (OLSR) scheme, and an application specific scheme. The study's focus compared the average number of retransmission of REM packets (overhead) transmitted per node versus number of nodes in the network. Although measuring retransmission provides insight into extenuated overhead, the study failed to address the effect of REM exchange to the average throughput of non-control data on the network or the effects to system packet delay. This research document intends to investigate this missing information.

2.4 Background Summary

This chapter presented research on issues inherent in cognitive radio development. Specifically, it addressed the current state of rendezvous algorithms based on scenario applicability, and presented known research in evaluation of network performance overhead costs caused by implementing DSA. This study of previous related works revealed an oversight in research for methods for rendezvous capable of operation under both SPSFH and FFH. Additionally, it uncovered a lack of research into the the overhead requirements involved with exchanging REM data on a constant contact network.

III. Methodology

This chapter is divided into two experimental sections to assist in organization and presentation. The first section addresses an FPGA implementation of Gold's algorithm. The section begins by summarizing the development methods and initial testing of the design. It then presents a test methodology for implementation of the hardware design. The methodology describes, in a systematic way, the process of investigating the implementation's resource requirements and timing restrictions. The second section of the chapter presents three different generalized REM exchange protocols for investigation, and the proposed experiment. The experiment specifically examines the effects of the REM exchange protocols on network overhead costs in relation to network scaling and performance.

Through the above development of a hardware system and the experimentation, this research investigates five questions about FHSS DSA cognitive radios:

1. What are the hardware requirements and timing restrictions for implementing Gold's algorithm on an FPGA based testbed?

2. To what extent does coordination of radio networks through hop sequence estimation allow for quicker rendezvous versus current method of key synchronization?

3. How does hop sequence estimation perform within the DSA paradigm regarding rendezvous compared to current methods?

4. How do different REM exchange protocols affect the network overhead costs in relation to network scaling and performance?

5. To what extent does network size limit the time interval between REM exchanges?

3.1 Hardware Implementation of Gold's Algorithm

3.1.1 System Development.

A testbed is created to test the effectiveness of Gold's algorithm. The design uses the Xilinx ML507 development board, which does not have the ability to create radio frequency transmissions. However, it does possess the ability to record and play audio. As such, the audio peripheral is used instead of RF as an analogous medium [40]. An FPGA platform is chosen for implementation for two reasons. First, FPGAs offer operation speeds capable of supporting the computation level requirements of FHSS. Secondly, they offer support for the rapid prototype development required by research. Note that all current AFIT research efforts on CR are directed toward Rice University's Wireless Open Access Research Platform (WARP) which is an FPGA-based system [1].

Figure 3.1: Top-level system architecture

The testbed design consists of both a transmitter and a receiver. Figure 3.1 presents the top level architecture of the system. The receiver will contain the implementation of Gold's algorithm. The transmitter design uses an LFSR with variable bit length to create its hopping sequence. The bit length, L, as well as the bits selected for feedback determine the period associated with the hop sequence. The capabilities and properties of LFSRs have been well studied, and selection of feedback bits to create the longest period is beyond the scope of this paper; however, it can be stated that the longest sequence period possible of any LFSR is equal to $L^2 - 1$ [27]. A representation of an LFSR PRNG is shown in Figure 3.2. During operation, several bits within the LFSR are statically selected

as *tap* bits. The maximum number of operating frequencies, N, of the system determine the number of tap bits needed where $N = 2^{\#taps}$. The tap bits control the audio frequency (equivalent to RF channel) used for transmission.

The receiver serves as the acquisition device. Functionally, it listens for the audio tones of the transmitter, estimates the sequence using Gold's algorithm, and finally synchronizes with the incoming sequence. To perform these actions, a data flow occurs. First, an audio microphone collects the signals produced by the transmitter. This detected audio passes to a Fast Fourier Transform (FFT) core that translates the signal into the frequency domain. A comparator is attached to the output of the FFT core to detect when a single selected frequency is present. The presence or absence of the frequency detection signal represents the communication on a carrier frequency in RF. This detection signal serves as input into the Gold's algorithm IP core. The IP core detects the time of arrival differences for the frequency, calculates the state of the transmitter's LFSR, and determines the appropriate tap bits to replicate the incoming sequence. Both the calculated LFSR state and tap bit locations pass to the receiver's LFSR. Upon successful estimation of the hop sequence, the receiver produces verification of synchronization.

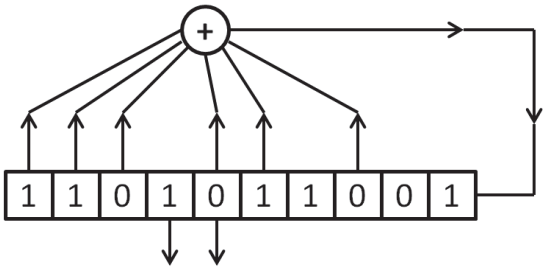


Figure 3.2: Example Source LFSR PRNG

3.1.1.1 Gold's Algorithm VHDL Design.

The algorithm IP core consists of seven distinct computation processes, or blocks, and a top-level wrapper for component coordination and control. Each block corresponds

with a phase of the matrix computations implemented within Gold's algorithm. Appendix B provides a detailed description of each block's implementation in VHDL. Appendix C contains the VHDL code for the algorithm core.

3.1.1.2 Initial Design Testing.

Determining hardware requirements and timing restrictions first requires verification of the algorithm's functionality when implemented in the design described above. As such, a reasonable initial test is generated to determine the operation of Gold's algorithm. The test implements a radio pair capable of frequency hopping over 32 frequencies according to a uniform distribution. This test therefore utilizes a 16-bit length LFSR with five feedback taps and five frequency taps. The maximum sequence period for a 16-bit LFSR is 65,535 cycles. The use of audio tones and the FFT limit the frequency hop rate. For demonstration purposes, the hop rate is set to 6 Hz or six frequencies per second. A successful demonstration is defined as proper function of the algorithm to synchronize the receiver to the transmitting system. For system testing, it is assumed that no interference tone or noise generators are present.

The scenario described ensured proper operational testing of the testbed design. Upon implementation, the receiver proved the proper operation of Gold's algorithm to synchronize with the transmitting audio stream. The test was repeated for different transmitter starting states, feedback bits, and tap bits parameters. This initial testing verified that the algorithm IP core functions as expected.

3.1.2 Test methodology for Experiment 1 - Gold's algorithm.

This section establishes the testing methodology for determining Gold's algorithm's hardware system requirements and timing limitations.

3.1.2.1 Parameters/Factors/Levels.

Implementation of the algorithm depends on several parameters for system design. Some parameters significantly affect the hardware resources required for implementation

as well as the maximum clock frequency for stable operation. These significant parameters define the design factors, and are varied in this experimentation. Understanding the resource requirements enables designers to properly select components based on desired radio capabilities while the hardware design's timing limitations restrict the operational speed of the algorithm. The following parameters are considered in this experiment:

1. Frequency hop rate – defined as the speed at which frequency changes occur, measured in hops per second. The maximum value for this parameter is limited by the maximum operation speed of the algorithm itself when implemented in hardware; however using modern processors the reaching this rate is unlikely. The frequency hop rate is determined by the target device and remains constant. The rate does not directly affect the required hardware resources or timing constraints, and therefore is not to be considered a factor. This rate must be known for Gold's algorithm to function.

2. LFSR bit length – Bit length for the LFSR are selected as part of system design requirements. As such, the length significantly affects the resource requirement needed for implementation. The LFSR bit length is derived from the desired sequence interval of the radio system where,

$$\text{Sequence interval} = \frac{\text{Sequence period}}{\text{Frequency hop rate}}$$

   Proper selection of the feedback bits can produce a maximum sequence period of $L^2 - 1$. For military applications, it is common to have the requirement that the sequence interval be greater than 24 hours so that the hop sequence does not repeat over within one day. Eight levels of bit length variation are examined: 8, 12, 16, 20, 23, 28, 32, 36. These levels allow sequence periods of 255 to 68.719 million cycles. Using a sequence interval requirement of 24 hours, these levels correspond to a hop rate range of one hop every 338 seconds to 795,364 hops per second. This wide

range covers known operational hop rates of modern systems. Four bit spacing between levels is chosen to explore FPGA operations that allocate partial resources such as using only 4-bits of an 8-bit register.

3. Feedback bits – Although feedback bits determine the period of the sequence, the implementation design presented allows any combination of feedback bits to be selected from the LFSR at system startup in software. As such, any change in the selection of feedback bits does not affect the hardware resources or timing constraints. This parameter is not considered a factor for this experiment.

4. Operational frequency range (Tap bits) – the maximum number of frequency channels of the system determines the number of possible tap bits, $N = 2^{\#taps}$. For proper operation, the receiver must have foreknowledge of the number of taps bits used by the transmitter. The variation in taps significantly changes the logic requirements of the system. Therefore, the number of tap bits affects the resource requirements and timing limitations, and is considered a factor for this experiment. Seven levels for the number of tap bits are examined: 4, 5, 6, 7, 8, 9, and 10. This selection allows for systems utilizing from 16 to 1024 distinct frequency channels.

Table 3.1: Summary of Parameters/Factors for Gold's algorithm experiment

Factor	Levels
Frequency hop rate	6 Hz
LFSR bit length	8, 12, 16, 20, 24, 28, 32, 36
Feedback bits	Constant in hardware (set in software)
Tap bits	4, 5, 6, 7, 8, 9, 10

3.1.2.2 System Metrics.

System metrics for this experiment focus on the resource requirements to implement the Gold's algorithm IP core, and timing limitations imposed by the implementation.

Implementation of a design to a FPGAs is both software and device specific. For this experiment, Xilinx ISE v13.2, application O.61xd, synthesizes a netlist to target the Xilinx ML-507 development board, revision A, which utilizes a Virtex-5, version xc5vfx7dt, FPGA. Xilinx ISE v13.2 uses the Xilinx Synthesis Technology (XST) process by default, but is capable of supporting the 3rd party synthesis tools of Precision from Mentor Graphics Inc. and Synplify from Synplicity Inc. The default synthesis tool will be used.

This software produces a detailed report as part of the synthesis process which include a listing of used resources. This report provides resource information at three stages during synthesis as well as a final device utilization summary. The first stage's report covers the initial Hardware Description Language (HDL) synthesis before any optimization occur. This stage is not useful since it contains many unneeded connections and components. The second stage covers the advanced HDL synthesis. This stage's report presents the system resources needed in terms of macro statistics after a high level optimization. The final stage covers the low-level optimization, and the report addresses specific cell usage. The device utilization summary provides a global view of the resources used for synthesis in terms of slice registers and slice Look-up tables (LUTs). A slice is the principle programming resource unit of an FPGA. A single slice consists of both LUTs and Flip-Flop/Latches. The number of each on a single size varies depending on the FPGA. As such, this utilization summary is specific to the Virtex-5's onboard resources, and cannot be applied to all FPGAs; however, examination of utilization will show general resource trends. Both the utilization summary and the advanced HDL synthesis report will be examined and compared in this experiment. The Xilinx XST synthesis report also addresses timing. The report provides expected timing restrictions of the VHDL design

implemented. Specifically, it establishes the minimum clock period, maximum clock frequency, minimum input arrival time before clock, maximum output required time after clock, and maximum combination path delay for an implemented system. Of these, this experiment collects and analyzes the maximum clock frequency. This value represents the highest clock frequency the algorithm can be utilized on the Virtex-5 to ensure stable operation. Note that this value is **not** equivalent to the maximum frequency hopping rate.

3.1.2.3 Expected results.

FPGA implementations for designs typically require set resources for the complexity of a system. This experiment investigates the several configurations of a working design. As such, the hypothesis of this experiment is that the resources required to create the working implementation of Gold's algorithm can be estimated given LFSR bit length and the number of tap bit. Additionally, an estimate of the the maximum clock speed should be calculable using the same parameters. The estimations are based solely on using the Xilinx XST synthesis tool and the Virtex-5 targeted device. Although other devices may differ in results, one device under test should provide insight into implementations on other devices.

3.2 REM Exchange Overhead

In a network of radios, a universal view of the environment must exist for radios to coordinate changes. Different from the multi-dimensional REM concept presented in Section 2.3, this experiment for simplicity utilizes a simple logic vector. This logic vector represents each communication channel as available or unavailable using binary 1's or 0's respectively. Each radio must coordinate its local REM with every other radio in the network to ensure only common available channels are utilized. Figure 3.3 shows the coordination of multiple nodes to create a shared REM. The dark sections represent unavailable channels, and light sections represent available channels. The resulting vector from coordinating multiple REMs across a network is the master REM or network REM.

Analyzing the methods of sensing the environment to create a node's REM is outside the scope of this research.

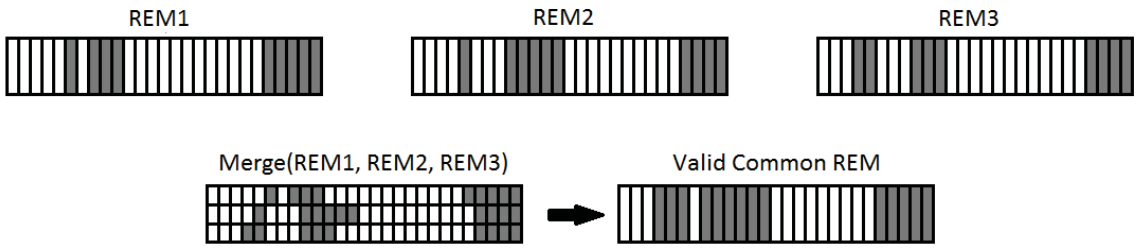


Figure 3.3: REM Merger [22]

After a system rendezvous occurs, the system must coordinate the information required to maintain a stable connection. The specifics on exchange or distribution are rarely described, and seem to be not well researched. As such, this thesis examines several exchange protocols for transferring control or REM data among radios to determine the expected overhead costs and scalability of the network. This thesis identifies and tests three possible exchange protocols for a constant contact network.

The simulation models implemented in this research are modifications of the IEEE 802.11b Wireless Local Area Network (WLAN) computer communication specification. WLAN, being one of the most widely used standards in the world, was selected as the best starting point due to the readily available OPNET simulation wireless node models, source code, and documentation. As background, IEEE 802.11b contains two primary MAC coordination functions. Distributed Coordination Function (DCF) is the basis of the standard CSMA/CA access scheme. For DCF, a transmitting node first listens to the medium to determine if any other nodes are transmitting. To negotiate for the medium with other nodes, a contention window is created. This contention window represents the time in which the medium can be taken control of by a single node. To avoid colliding during transmissions, each node selects a random backoff interval within the contention

window to stall before attempting a transmission. The first node to complete its backoff time and determine that no other node is transmitting may take control of the medium. DCF is considered the normal operating condition for WLAN [12].

The second primary function is the Point Coordination Function (PCF). PCF provides a contention-free service to exchange information. The access point of the network in IEEE 802.11b initiates the PCF period by using a broadcast beacon. Once initiated, the access point controls the wireless medium. It then requests and sends data as needed. The PCF period lasts until the access point relinquishes control, or a time limit is reached. DCF operations start once PCF period ends. The existing implementation of these functions in the OPNET wireless nodal model (802.11b) are leveraged to implement the three exchange protocols for determining the impact of REM exchange overhead on user, non-control, traffic.

These function are the foundations for the OPNET simulations models used in this experiment. All network activity using WLAN is performed in-band since commercial devices rarely contain more than one transceiver.

3.2.1 REM exchange methods and model design.

This experiment examines the affects of three general exchange protocols for transferring REM information among the network.

3.2.1.1 Polling Protocol.

The Polling protocol is a centralized network design based on IEEE 802.11b's PCF. Polling requires a single radio to be assigned the position of 'controller' or 'master' for each exchange. Theoretically, this control radio is determined through one of many methods to include longest living member of the network, based on MAC serial number, or even the most powerful radio in the system. For this research, the control radio is selected prior to simulation. In this protocol, the control radio coordinates the local REM from each radio in the network by leveraging the polling functionality of the PCF. Upon

protocol initiation to take control the medium, each radio in the network is 'polled' by the controller as a request for the radio to send its local REM. All radios in the network are polled individually during this modified PCF period. The control radio then computes the master REM for all other radios. After computation, the control radio broadcasts the master REM to the members of the network. Figure 3.4 graphically depicts the operation of this method. Once all nodes receive the master REM, the network switches back to normal operation using the DCF.

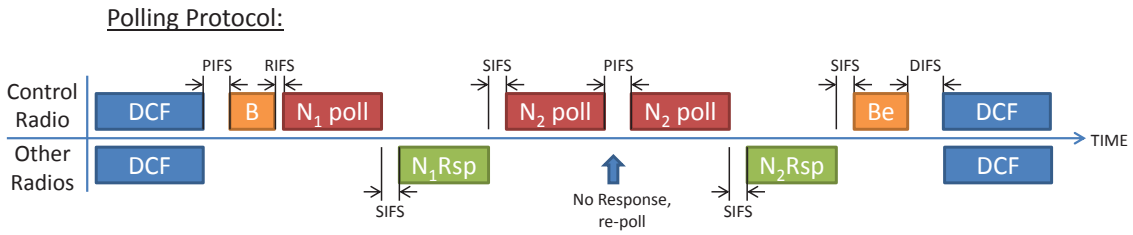


Figure 3.4: Polling Protocol for REM Exchange

3.2.1.2 Time Division Protocol.

The Time Division protocol focuses on a distributed network design where each radio must communicate with all other radios in the network. Each radio collects the local REMs of each neighboring radio and calculates the master REM for itself. This method assumes that 1) every radio is within range, 2) every radio uses the same master REM algorithm, 3) each radio interprets the usage of the master REM the same way, and 4) a pre-coordinated execution time exists within the network. Figure 3.5 graphically depicts the operation of this protocol. Time division is a common method of coordinating a communication medium. For this protocol to work in REM exchange, each node must know its precise order within the network to determine when to transmit. The Time Division protocol also utilizes a control radio, but this radio only initiates the exchange

period. For simulation testing, the node order schedule is determined by their node identification number.

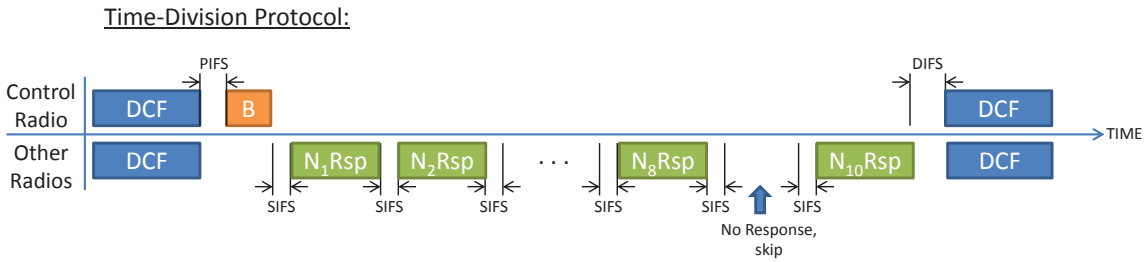


Figure 3.5: Time Division Protocol for REM Exchange

3.2.1.3 Exponential Backoff with Priority Contention Protocol.

Exponential Backoff with Priority Contention Protocol, referred to as simply the Priority protocol, presents a reduced communication protocol to exchange REM information among network radios. As with the other protocols, a control radio is selected among the radios. The exchange Priority protocol works in three stages. First, the control radio will distribute a Suggested Master REM (SMREM). The SMREM consists of the controller's local REM along with any other known restrictions/predictions to the environment. In Stage 2 each non-control radio compare their local REM to the SMREM. If a radio wishes to further limit the SMREM (i.e. a suggested open channel is detected as unavailable), the radio updates the control node with the 'dispute' information. This method functions similarly to DCF operations except that only REM information is exchanged and a priority backoff system is used to maximize the reporting process. The selection of which radio has priority to respond to the control node is performed by reducing the contention window of each node as a percentage of disputes. Each node adapts its contention window using the following equation.

$$\text{New Window} = \text{Old Window} \times (1 - \frac{\text{disputes}}{\text{Size of REM}})$$

The control radio incorporates all received disputes transmitted from network radios. The control radio waits for maximum elapse time to occur after a dispute to indicate all disputes were reported. After the elapsed time, the control radio performs stage 3 which posts a final master REM to be utilized by the network. Figure 3.6 graphically depicts the operation of this method. A key attribute of this protocol is a non-control radio's ability to listen to the medium during this exchange period. This allows a radio to detect if another radio reports a dispute which covers the listener's disputes. If this occurs, the radio no longer needs to report the dispute. In this protocol, it is possible that two nodes select the same backoff period, and attempt to transmit at the same time. If this occurs, the transmitting have no immediate indication that a problem occurred while all other nodes will have detected the collision. These transmission error nodes will discover the problem only if another node posts an dispute after the collision occurs upon which they will queue another dispute response.

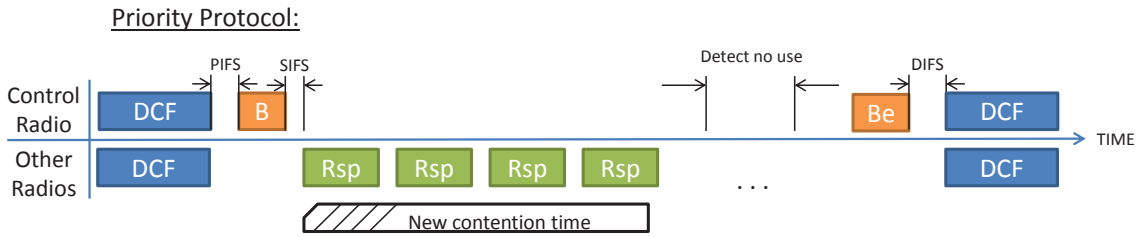


Figure 3.6: Exponential Backoff with Priority Contention Protocol for REM Exchange

3.2.2 Test methodology for Experiment 2 - Network overhead.

3.2.2.1 Experimental goal of simulation.

This simulation generates data used to evaluate the three methods of REM exchange protocols explained above. In particular, the data is used to determine how the different REM exchange protocols affect the overhead cost in relation to network scaling and performance. Overhead data is the necessary exchange of environment data (REM data)

distributed between a group of nodes to facilitate future communication. This experiment utilizes a simulated wireless network for evaluation of overhead data over many network configurations. For these experiments, a network is considered 'degraded' when throughput of normal data traffic (non-REM data) is reduced by a threshold of 10%. This level is chosen due to the level of data loss acceptable for Voice over IP operations.

3.2.2.2 System Under Test.

The system under test consists a virtual network created using the Riverbed OPNET modeler suite, version 15.0.A.PL1(build 8165). The network comprises of a number of wireless nodal models designed to emulate the behavior of a DSA radio. A complete nodal model contains the entire seven layers of the Open Systems Interconnection (OSI) framework model (ISO/IEC 7498-1); however, to limit the external influences of each layer, only the Network, Link, and Physical layer are implemented within the models of this experiment. Each node transmits and receives both normal network traffic and REM overhead data traffic; however only the amount of normal network traffic (non-control traffic) is evaluated within the collected metrics. The transmission characteristics of each node is modeled using OPNET's IEEE 802.11 wireless suite. These models utilize a radio transceiver pipeline consisting of a 13 step sequence to evaluate transmission delay, antenna gains, propagation delay, background and interference noise, error detection and correction, and signal-to-noise ratio requirements for transmissions between any two nodes. For simplification of the problem space, each node is assumed to be within transmission range of all other nodes within the network. All nodes in the network remain stationary, and are located within a 100m by 100m grid. This research excludes any node mobility due to the added scenario complexities that would be required. The grid shape and dimensions were initially selected as a continuation of past research to create an RF environment test set. Although the test set results proved not applicable for this experiment, the layout ensures a consistent design for the simulated network.

The REM exchange protocol used is the component under test. For every simulation setup, each node in the network uses the same nodal model with the only distinction consisting of the selection of a master or control node among the nodes. Table 3.2 presents the common critical OPNET attributes shared by all nodes. The transmission start time describes when each node will begin attempting to transmit non-control data traffic. This allows for spacing in the beginning of the simulations. Packet size for non-control data is one parameter that directly affects the workload put on a network. This attribute is set so that the average packet size is a fixed value among the network. The other attributes help define characteristic features of the system under test. All the values presented are common throughout all simulations in this study.

Table 3.2: Node Attribute Settings

Attribute	Value
Transmission Start time (sec)	exponential(1)
Packet Size (bytes)	exponential(1024)
Data Rate	1 Mbps
Transmit Power	0.1 W
Retry limit	7
Buffer Size	256000 bits
Maximum Transmission Unit	1500 bytes
Destination address	Random

3.2.2.3 Parameters.

The term *parameter* refers to any setting or environment variable that affects systems performance. These parameters are divided into those specific to the system or to the workload. System parameters change between different simulations, but remain constant within a single simulation. The amount of user specified non-control data traffic placed on

a system defines the workload parameters. Workload parameters can vary during a single simulation over time, and vary for different users. The following list defines the system and workload parameters for the system under test.

- REM Exchange Protocol– The protocol under test that explains how the exchange of REM information is executed among the nodes in a network.
- Network size – the network size describes the number of simulated nodes within a simulation. Since each node is capable of transmitting data, every additional node increases the overall data workload as well as the amount of overhead data required for a network to coordinate REMs.
- REM Update Interval – The time interval between initialization of the REM exchange procedure. This interval can occur at a set interval, or vary based on the performance or state of network operation. As the interval decreases, the amount of overhead data required to be processed by the network increases.
- Node Separation – the distance between any set of nodes. This experiment limits all nodes within a 100m by 100m grid. The nodes are placed 10 meters apart (vertically and horizontally) to create 100 possible locations (10 by 10) grid. The nodes are placed in a growing square pattern.
- System Channels – A DSA radio system is limited by the spectrum its transceiver can operate. The system bandwidth is divided into operating channels according to the transmission protocol utilized. For instance, a home network router operating according to IEEE 802.11 in North America operates in the 2400-2483.5 MHz frequency range. This bandwidth is divided into 79 maximum possible transmission channels for the standard. DSA coordination among radios will limit which channels are considered usable. The maximum channel number dictates the length of the REM required to represent the bandwidth. The simulations within this

experiment utilize a constant of 2048 possible channels to cover a larger range of possible channels. This constant length of the REM creates a constant packet size for control traffic used in the REM exchanges. Evaluating cross channel interference is beyond the scope of this research, so each channel is assumed to be non-overlapping and non-interfering.

- System Data Rate – the system data rate is the maximum rate at which data is transferred within the system, typically measured in bits per second. This rate impacts the time required to transmit and receive data transmissions thereby affecting the system's performance. As shown in Table 3.2, all nodes use a data rate of 1 Mbps.

- Data Workload – the data workload is a measure of the amount of normal (non-control) data traffic on the network. With OPNET, data workload is attributed to the packet size and interval of arrival of data from a higher OSI layer to the MAC layer. Both size of packets and the interval between packets can be modeled according to random distributions or set as a constant value within OPNET. These parameters impact the amount of normal traffic traversing the network. As displayed in Table 3.2, all nodes are modeled using a data generation interarrival time with an exponential distribution and an average packet size of 1024 bytes. The interarrival rate between packets is considered a factor in this experiment, and is discussed in the next section.

- REM similarity – the performance of an adaptive REM exchanging network can vary with the similarity of local REMs between the nodes. This similarity between REMs is based on the environment detected by the radios, and the threshold used to determine the presence or absence of a signal on any channel. Similarity can be attributed to locality to interference sources and node separation.

3.2.2.4 Factors.

Factors are a subset of the parameters that are varied in the experiment. Any parameter not considered a factor remains a constant throughout the experiment. The following factors varied within this study.

- REM Exchange Methods – This experiment evaluates three REM protocols: polling, time division, and exponential backoff with priority contention. All three are tested individually for a performance comparison.

- Network size – the network size will be varied to represent a network size of 4 to 100 nodes. As this number range creates a larger number of trials, the following levels are chosen for this study: 4, 9, 16, 36, 49, 64, 81, and 100. These eight levels allow for each trial to progressively grow in the number of nodes while maintaining node separation intervals. Each network size forms a square grid resulting the following configurations: 2x2, 3x3, 4x4, 5x5, 6x6, 7x7, 8x8, 9x9, and 10x10.

- REM Update Interval – The intervals selected for this experiment are selected specific to each network's size. Four or five test intervals were selected for each network size. The number of nodes in a network directly affects the minimum interval required for the network to function. The minimal interval is the maximum time required to execute a REM exchange. Any interval smaller than this minimal value produces a network condition where no non-control data would be transferred resulting in an unusable network. A pilot study was performed to approximate the needed values. The pilot study is further discussed in Chapter 4. Table 3.3 displays the interval values used in the experiment based on network size.

- Data Workload – the data workload, like the REM update interval, is different for each network size. The values selected for the experiment were determined from the pilot study of a baseline network (a network operating without performing a REM

Table 3.3: Test Levels for REM exchange intervals

Nodes	REM Update Interval (sec)
4	0.25, 0.5, 0.75, 1.0
9	0.25, 0.5, 0.75, 1.0
16	0.25, 0.5, 0.75, 1.0
25	0.25, 0.5, 0.75, 1.0
36	0.5, 1.0, 1.5, 2.0
49	0.5, 1.0, 1.5, 2.0
64	0.5, 1.0, 1.5, 2.0
81	0.5, 1.0, 2.0, 3.0
100	1.0, 2.0, 3.0, 4.0, 5.0

exchange). The identification of the peak data throughput for each network size in the pilot study allows the experiment to narrow the number of test value. The values in Table 3.4 represent the inter-arrival rate at each node to be used in the experiment. The values are used as the average to an exponential distribution function.

- REM similarity - No viable existing data was found during review; therefore, characterization scenarios were investigated based on different levels of REM similarity. Four generic situations were created to provide bounding of the problem.
  - In the first scenario, referred to as the Unique Scenario, all nodes contain at least one unique channel that must be disputed (upper bound).
  - In the second scenario, referred to as the Identical Scenario, all nodes contain the exact same available channels (lower bound).
  - In the third scenario, referred to as the Random Scenario, every channel in a local REM is randomly selected as available or not available with a 50% probability.

Table 3.4: Test Levels for Nodal Data Workload

Nodes	Packet Interarrival Rate (msec) per node
4	20 to 150 with step size 10
9	100 to 200 with step size 10
16	200 to 300 with step size 10
25	340 to 450 with step size 10
36	500 to 800 with step size 20
49	720 to 1100 with step size 20
64	900 to 1800 with step size 50
81	1200 to 2300 with step size 50
100	1600 to 2100 with step size 50

– Lastly, a single scenario, referred to as the Real World Scenario, is constructed to represent conditions found in the real-world operations. A simplified representation of this scenario's setup is shown in Figure 3.7, and described below. The purpose of creating such a scenario is to establish plausible situations that may be encountered by a working network.

The Real World Scenario consists of an array of interference towers surrounding the network under test. Each of the 40 transmitter towers in Figure 3.7 represent a cluster of ten separate interference sources at the specified location for a total 400 transmitters. Each source within a cluster is characterized by an activity setting parameter, an interference range parameter and its location (the location of the cluster). The activity setting is the probability of being an active transmitter. Specifically, the ten sources operate with a 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, or 100% channel occupancy rate. No transmitters in the same location use the same probability (i.e. each uses a different rate). At the beginning of each simulation, each

transmitter will randomly select an interference range with a value from 0 to 140. This distance along with the location of the transmitter determines which radio nodes are affected by each transmitter. The colored circles in Figure 3.7 are an example of the interference range of three sources.

This Real World Scenario accounts for both spectrum variation over time, and locality to interference sources. As such, it provides a reasonable set plausible RF environments for the simulations. It is possible for a tower to affect all nodes, no nodes or some nodes. As an added complexity, the design pairs sources on either side of the grid to share frequencies (eg. a tower on the north end uses the same channels as on on the south). Each source is assigned a range of 10 channels to cover 2000 frequencies between the paired towers. The 48 remaining channels represented in a REM are considered always available. This design simulates a larger region where the outlying radios may have different interference sources that operate on the same frequency channel. At the start of each execution of the REM exchange protocol, every node in the network builds its own local REM based on the condition of the interference towers and its proximity to those sources which are active. If the node falls within the interference range of an active transmitter, the 10 frequencies controlled by that active tower will become unavailable.

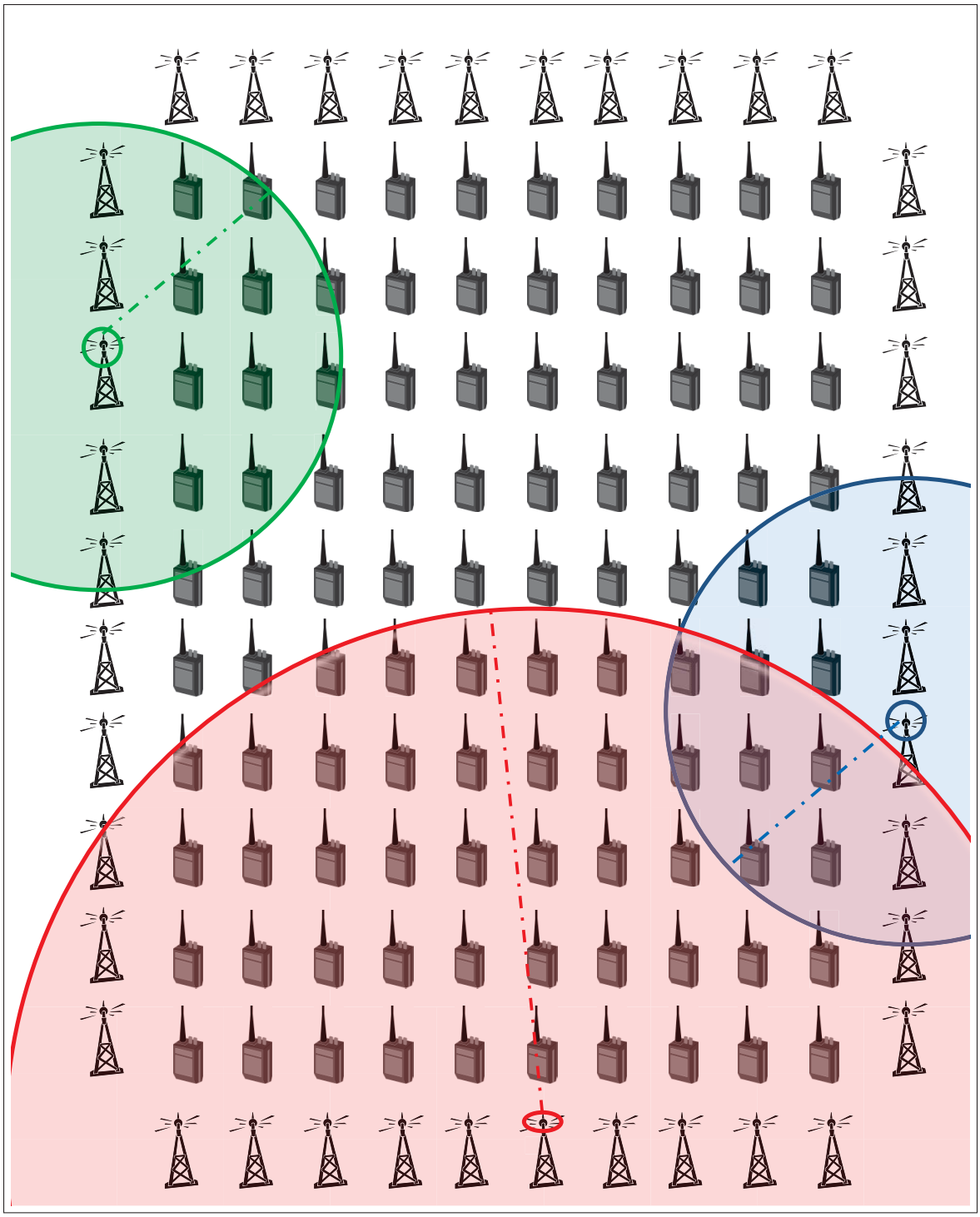


Figure 3.7: Real-World Scenario

3.2.2.5 Performance Metrics.

The OPNET simulations report five network performance metrics for analysis. Each metric provides insight into the functionality and costs associated with the REM exchange methods. All metrics are measurements of the normal data (non-control) traffic moving through the network.

Network Throughput - For this experiment network throughput is defined by data generated, transmitted, and successfully received by the intended destination over time. This metric represents only normal data traffic, and does not include the overhead of data exchanged for the REM. For each simulation, a control node initiates the REM exchange procedure. Each other node responds according to the protocol designated for the simulation. Any packets transmitted for the purpose of coordinating REM data, including the process initiation packet, will be considered an overhead packet. Network throughput is measured in bits per second.

Data dropped due to buffer overflow - Each node is capable of queuing data before transmission. Data packets arriving to the MAC layer of a node from the node's internal packet source must enter the queue before transmission. The hardware of a radio transceiver limits the size of queue. Overflow occurs when the queue is at maximum capacity, and the packet generator pushes the packet to the MAC layer. In such a case, the new data packet is discarded by the MAC layer. The amount of data dropped due to buffer overflow indicates a network's ability to transmit the packets generated by each node. Data dropped is measured in bits per second.

Data dropped due to exceeding retry limit - A wireless network contains the risk of more than one node attempting to transmit data at a single time. This event is called a transmission collision. If this occurs, each node can attempt to retransmit the packet at a delayed time. A set maximum number of retries may be attempted before the data packet is discarded by the MAC layer. The average amount of data dropped due to this retry

threshold is examined in this experiment to track the rate of successful delivery of network traffic. Data dropped is measured in bits per second.

Retry attempts - in addition to data loss, the number of retransmission attempts to successfully deliver a packet indicates the performance of a network. An increase in retransmission attempts signifies a highly congestive network and can lead to unacceptable performance.

Packet Delay - Measured in seconds, delay is defined as the time from packet creation by the packet generator to the time the packet is successfully received at its destination. Significant packet delay impacts the usability of a network for time dependent transmissions.

3.2.2.6 System Workload.

For each simulation, a traffic load of normal data is generated for the network to process. This workload is a function of the network size and the workload generated by each node. The system workload holds constant for each simulation, but varies among batches of simulations to characterize a networks ability to handle differing traffic loads. The maximum load is defined as the rate of traffic generation that produces the peak maintainable throughput rate for a network configuration. From the pilot study presented in Appendix A, the throughput rates that occur at the maximum load points are negatively correlated to the size of the network.

3.2.2.7 Experimental design.

A complete factorial design is used for this experiment. Each combination of method, network size, traffic loads, REM update interval, and scenario is executed as a single simulation. Varying the traffic load creates a simulation batch, and multiple batches are executed for each required scenario. The Polling protocol and the Time-Division protocol each require only a single scenario for characterization, while the Priority protocol requires the four distinct scenarios presented above. Each separate simulation is

to be repeated 15 times utilizing a different seed value. The varying the seed value (values 1 to 15) changes the outcome of OPNET's internal random number generators used for both traffic load parameters and the REM creation of the Random Scenario. These multiple runs allow the establish a 95% confidence interval for analyzing the collected performance metrics. Table 3.5 displays a matrix of factor levels for the execution of a single scenario. The testing of each scenario consists of 8,385 simulations. A total of 50,310 simulations are performed to characterize the three REM exchange protocols.

Table 3.5: Test Matrix of simulation runs

Network size	Data Workload	REM update intervals	Seeds values	total simulations
4	14	4	15	840
9	11	4	15	660
16	11	4	15	660
25	12	4	15	720
36	16	4	15	960
49	20	4	15	1200
64	19	4	15	1140
81	23	4	15	1380
100	11	5	15	825
Total	–	–	–	**8385**

3.2.2.8 Expected results.

Considering the number of transmissions required by each node, the length of REM, and system workload, it is reasonable to hypothesize that the Priority protocol for REM exchange will prove to have the highest scalability among the protocols tested determined by a 10% degradation from a baseline network in regards to network throughput.

However, the Time-Division protocol is expected to provide the best performance for a small network. The Polling protocol and Time-Division protocol are expected to scale poorly for two reasons. First, the amount of overhead traffic is expected to increase linearly as the network size grows. Secondly, the pilot study demonstrated that network throughput diminishes as network size grows due to higher probability of transmission collisions. On the other hand, the performance of the Priority protocol is expected to easily adapt to larger networks since not all nodes are required to transmit information during each REM exchange. This hypothesis is formed from the expectation that more nodes in a network does not increase the amount of REM traffic transmitted. The ability to sense another node's disputes to the suggested REM allows this assumption.

IV. Results and Analysis

This chapter presents the results and analysis of two experiments. Section 4.1 details the results and analysis of implementing Gold's algorithm on an FPGA. Section 4.2 describes the results and analysis for the REM exchange simulation experiment.

4.1 Gold's Algorithm

An hardware implementation of Gold's algorithms was constructed to determine the timing constraints and resource requirements in various configurations. The experiment synthesized 53 configurations of Gold's algorithm for comparison. These configurations varied in LFSR bit length and the number of tap bits. This section first presents the timing constant results for the implementations, and then discusses the resource requirements. Lastly, the performance of Gold's algorithm as a rendezvous protocol is examined.

4.1.1 Timing Limitation.

The term timing limitations describes a restrictions caused by the routing and timing of logic gates within an FPGA design. As any signal flows through logic gates and through the routing between the gates, timing requirements must be meet to ensure proper signal propagation. During analysis of the different possible signal, the path that requires the most time to propagate through is defined as the critical path. This critical path is the limiting factor for determining the maximum clock frequency that a system can correctly operate. If the maximum clock frequency is exceeded during operation, the device may not function as expected, and report invalid results. The maximum clock frequency determined each tested configuration is displayed in Table 4.1. These timing results were an output of Xilinx ISE v13.2 using the XST synthesis process.

The fastest maximum clock frequency occurred in the least complex configurations while the slowest occurred in the most complex designs. Analysis of the timing data

Table 4.1: FPGA Maximum Clock Frequency (MHz)

		Number of Tap bits						
		4	5	6	7	8	9	10
LFSR bit length	8	218.866	223.564	218.150	218.866	X	X	X
	12	194.590	181.291	173.974	175.131	174.978	164.042	136.799
	16	153.445	150.966	148.170	148.324	149.880	146.306	136.388
	20	149.566	129.820	136.631	120.424	123.747	122.145	114.613
	24	149.388	133.672	127.714	107.411	97.286	104.373	101.968
	28	137.696	121.788	105.274	98.561	94.357	86.088	85.874
	32	129.182	108.507	86.633	86.483	87.047	75.792	75.672
	36	126.556	93.371	81.907	79.264	79.220	74.532	71.628

showed no linear or polynomial relationships between the two variables and the timing results. A generic trend showed that increasing either variable caused a decrease in maximum operational clock frequency. This was an expected trend for any hardware system. Upon further investigation into the synthesis output files, it was shown that the largest cause of timing is not the hardware elements themselves, but the routing of the intermediate signals between logic elements. For instance, in the synthesis of the 36-bit LFSR with 10 tap bits configuration, the timing restriction occurred in the Frequency Tap stage. This signal delay consisted of 19.9% logic delay and 80.1% routing delay. Therefore, improving the routing between elements of logic could increase the design's maximum clock frequency. The routing of the implemented design was determined by the Xilinx ISE's Map, Place and Route functions. These functions handle designs based on either timing-driven constraints or cost-based constraints. The above results were generated based on using timing-driven constraints; however, since no initial timing requirements existed during development of the design no specific timing restraints were fed to Xilinx to meet.

The timing restrictions collected represent the results produce by only one synthesis tool. Other synthesis tools may produce different results based on the exact placement and routing of logic gates and signals. Future work should consider extenuding this study to use other synthesis tools for comparison.

4.1.2 Resource Requirements.

The Xilinx v13.2 using the XST synthesis tool reports utilization data for the implementation of Gold's algorithm on the Virtex-5, version xc5vfx7dt, FPGA. The reports for each configuration include both register usage and LUT usage. A separate analysis is performed for each of these two types of resources.

4.1.2.1 Register Utilization.

An analysis of the synthesis reports showed the number of used registers in the design as a function of both the LFSR bit length and the number to tap bits. Figure 4.1 and Figure 4.2 show the resulting register usage by varying each parameter independently.

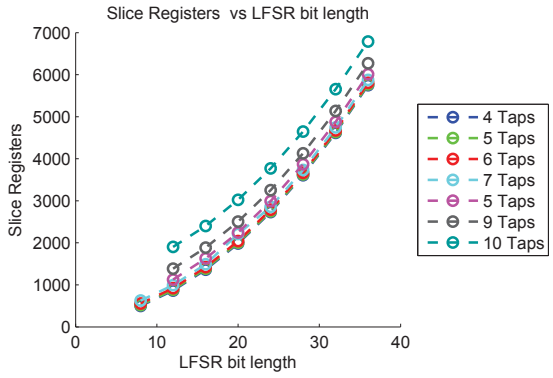


Figure 4.1: Register Usage vs LFSR bit length

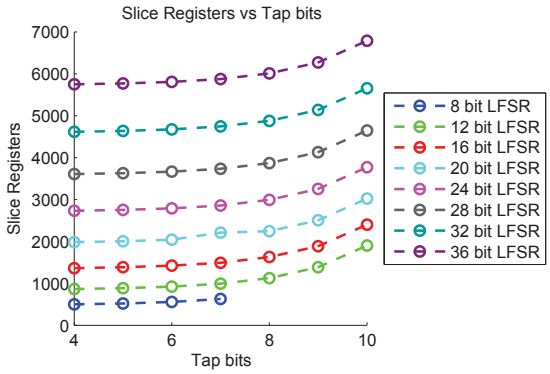


Figure 4.2: Register Usage vs Tap bits

The consistent shape of each line plot as a function of a single variable signified that the cross interaction between the variables is minimal. This is verified with through the use of MATLAB's curve fitting toolbox (v3.2.1). Independently the above charts were

approximated by the following two equations:

$$\text{Registers} = 4 \times L^2 + 11.4 \times L + C_1, \text{ where L = LFSR bit length}$$

$$\text{Registers} = 9.3 \times T^3 - 151 \times T^2 + 835.5 \times T + C_2, \text{ where T = number to tap bits}$$

The values, C_1 and C_2, represent the true interactions between the two variables. For simplification, any variable confounding was ignored to create a single simple equation to approximate register usage.

$$\text{Registers} = 9.3 \times T^3 - 151 \times T^2 + 835.5 \times T + 4 \times L^2 + 11.4 \times L + C_3$$

The value of C_3 was determined to be -1371 to produce the best fitting regression. For any regression, goodness of fit calculations determine the accuracy of the equation. MATLAB reported three factors in judging goodness of fit.

MATLAB reported a Standard Square of Error (SSE) of 2.109e+04 which is a measure of unexplained variation. MATLAB also reported R-squared value of 0.9999, and a Root Mean Square of Error (RMSE) value of 20.14. These value hold higher significance than SSE. R-squared, the coefficient of determination, is a measurement of goodness ranging from 0.0 to 1.0 where 1.0 is a great fit while 0.0 is a bad fit. This meant the approximation equation significantly defined the data points. RMSE corresponds to an unbiased estimate of error's standard deviation. Among the data points created by the different configurations, the equations estimated within 2% of the true value for all all except one outlying value.

4.1.2.2 LUT Utilization.

Figure 4.3 and Figure 4.4 represent the LUT usage of implementing Gold's algorithm. Although graphically similar to the register usage figures, the calculated regression equations to describe LUT usage were not consistent over the data. As such, no single form equation could be created to accurately estimate the hardware requirements in terms of LUT usage. From the data presented in the Utilization Summary, it was

determined that LFSR bit length significantly influences the number of LUTs over that of the number of tap bits. This influence is graphically shown by the steep exponential trend displayed in Figure 4.3.

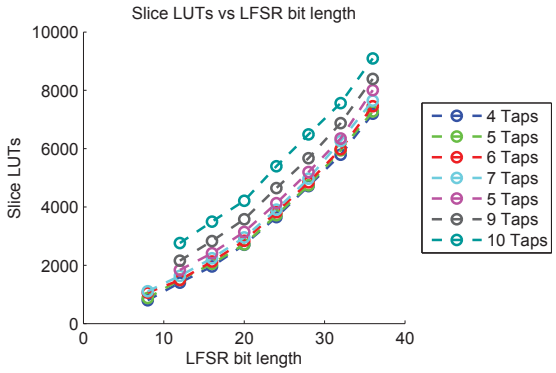


Figure 4.3: LUT Usage vs LFSR bit length

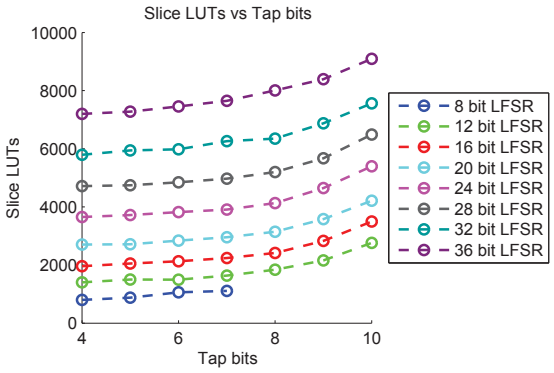


Figure 4.4: LUT Usage vs Tap bits

4.1.2.3 Other report results.

Examination of the Final Report's cell usage provided a breakdown of Basic Elements of Logic and Flip Flops/Latches required for implementing the algorithm. The Advanced HDL Synthesis Report displayed consistent resource usage across the different configurations. Table 4.2 summarizes the results of the Advanced HDL Synthesis Report in terms of Macro Statistics. These macro statistic values do not differentiate the size of each component. For instance, the multiplexer size for each configuration was based on the LFSR bit length utilized. A full chart with results from each synthesis report is included in Appendix A.

4.1.3 Rendezvous Performance.

The use of Gold's algorithm is not always the best choice when selecting a rendezvous scheme for FHSS networks. A comparison was performed on the estimated time to rendezvous for both COD method and Gold's algorithm. The COD method demonstrated a faster synchronization time when initiated early in the code cycle, and also

Table 4.2: Advanced HDL Synthesis Report Macro Statistics

Macro Statistic	Value or equation
Finite State Machines	1
RAMs	1
Adder/Subtractor	L+11 (minus 1 if L is multiple of 16)
Counters	5
Latches	L+1
Comparators	16 when $T \leq 6$, else 17
Multiplexers	2
XORs	$L^2 + 2 * L + 1$

when the system utilized a slow hopping rate. Figure 4.5 graphically represents a cognitive decision curve for selecting between the two schemes. The FPGA configuration with a 36-bit LFSR and 10 tap bits produced the slowest operational speed as shown in Table 4.1. The figure represents such a system when utilizing its maximum clock frequency of 71.628 MHz. To determine the optimal rendezvous scheme, a user or logic program must determine an operational hop rate and the time of day in which rendezvous is to occur. If intersection on the figure falls within the yellow shaded region, it is better to use Gold's algorithm. If the intersection falls within the green shaded region, the traditional COD method is the better choice. This boundary curve incorporates the average time of acquiring the needed samples for Gold's algorithm to function, the time needed to run the algorithm to determine the system parameters, and the time required to catch up to the sequence of the transmitter.

4.1.4 Effects of sequence interpretation for Dynamic Spectrum Access.

The current design of the testbed and implementation was created to verify the functionality of Gold's algorithm. Gold's algorithm was not designed to specifically

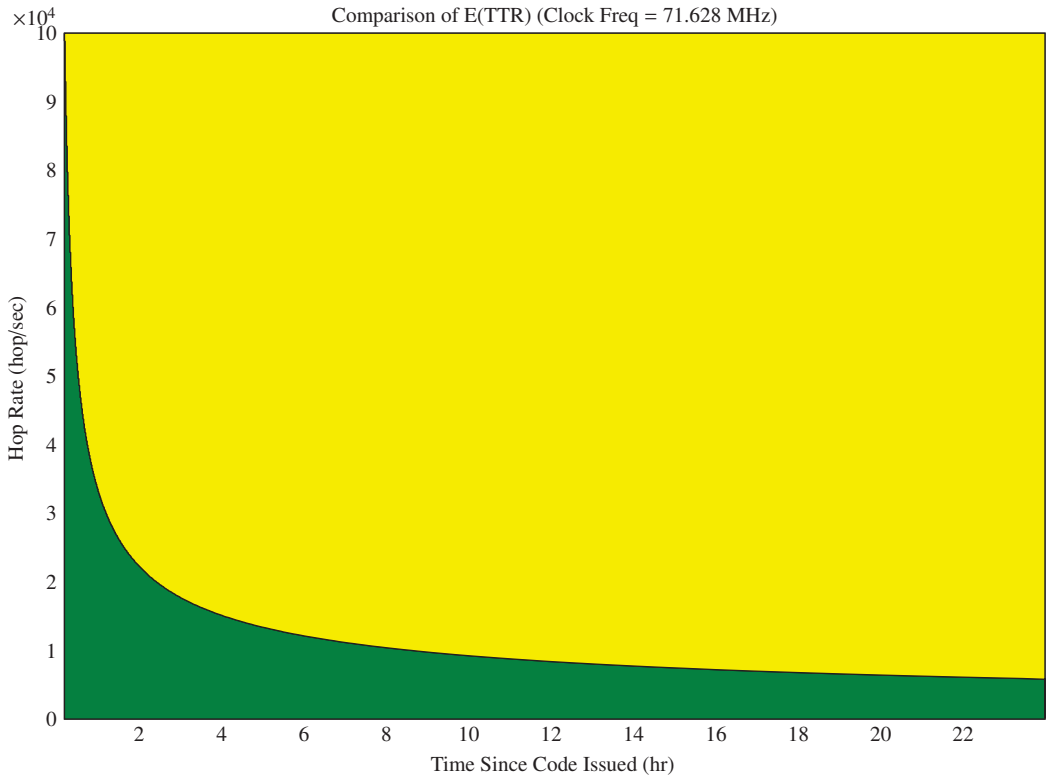


Figure 4.5: Decision chart for COD vs Gold's algorithm

function in a DSA environment. The application of DSA to FHSS functionally changes the performance and applicability of rendezvous algorithms. As such, different DSA incorporation methods are analytically examined in this section to determine their effects.

4.1.4.1 Skipping on channel unavailability.

The *skip* method describes the functional action taken by a MAC protocol when a frequency change is required and the next frequency is unavailable for use. For instance, imagine a system that uses an 8-bit frequency map. This system is capable of operating on any of 2^8 (256) frequencies. Let C_i be the operating channel of sequence S at time slot i. If currently in time slot i, the assigned channel for C_{i+1} could be any channel. If the desired channel C_{i+1} is not available, the system will simply move to the next frequency in the sequence, $i + 2$. This effectively removes channel C_{i+1} from S. This skip method has the benefit of maintaining both a constant network connection and its data rate. This

method, although valid, will break any implementation of Gold's algorithm since it effectively changes the time of arrivals between frequencies.

4.1.4.2 Hold on channel unavailability.

The *hold* method describes the functional action of not transmitting on an unavailable channel, and not modifying the hopping sequence. In other words, if a radio hops to a channel that is known to be unavailable, it will simply not transmit during the time slot. This holding of transmission will continue until an usable channel is selected. Gold's algorithm is compatible with this method. The time of arrival calculations are not affected by a failure to detect a signal (false negative), or absence of a signal (true negative). A network utilizing the hold method may experience reduced data rates. Note that if using FFH, a majority voting scheme can eliminate the data rate drop if unavailable channels in the sequence do not occur one after another.

4.1.4.3 Modulus of available frequencies.

The *modulus* method describes interpreting the hop sequence based on the number of known available channels. For example, say a radio determines that only 215 out of 256 operating channels are available. In this case, a frequency mapper re-interprets the next raw frequency value, such as channel 222, as a function of modulus 215. This produces $222 \mod 215 = 7$. Therefore, the system would hop to channel 7. This method remaps all possible outcomes of the a PRNG to the set of available channels. By this modulus operation, channel usage does not remain uniformly distributed over the available channels. Specific application of this method to Gold's algorithm requires thought into the application of the algorithm when no rendezvous is achieved. Using the modulus method generates the case where multiple PRNG values being assigned to a single channel. These creates a problem with the time of arrival calculations. Upon detecting that a single channel has multiple assignments, a protocol would be required to dynamically change the listening channel. This sets a requirement that $N/2 + 1$ channels must be available.

However, the modulus method remains a valid approach with this type of additional protocol.

4.2 Protocol Simulation

This section discusses the wireless network simulation results used to compare three possible REM exchange protocols. A total of 50,310 simulations were performed to characterize the three protocols. Each simulation requires an average of 20 seconds to execute resulting in a total run time of approximately 11.6 days. In this section, each protocol is compared to a baseline network's performance where the level of performance is evaluated using five metrics: network throughput, data dropped due to buffer overflow, data dropped due to exceeding retry limit, retry attempts and packet delay. These metrics were taken for normal (non-control) data traffic transmitted on the network. The section continues by presenting a series of representative examples in an attempt to condense the volume of data. Appendix A contains the full set of collected results.

4.2.1 Baseline Results.

A baseline assessment of the simulation network's performance was conducted for each network size over several data workloads. The baseline metrics were extracted from the pilot study and used to select several of the experiment factors. Figure 4.6 displays the network throughput for several network sizes over changing data workloads. This baseline throughput result is consistent with an exponential decrease in the peak average throughput as a function of increasing network size. The peak throughput achieved while using each of the exchange protocols will be compared to this decreasing exponential trend.

The baseline data also displayed a near linear relationship between the network size and the traffic workload required to achieved the peak throughput. Figure 4.7 graphically represents this interaction. Using MATLAB's curve fitting toolbox v3.2.1, the data in

Figure 4.7 produced a best fit curve estimate of

$$f(x) = 0.03254 * x^2 + 13.8 * x - 14.81$$

A goodness of fit evaluation of this fit curve produced a standard error of 10.31 and an adjusted R-square value of 0.9997 indicating a good representative fit. These consistent trends over changing network size make network throughput an ideal metric for determining the effects of REM exchange. Therefore, each of the three exchange protocols were compared to these baseline results to determine the overhead costs of distributing REM information.

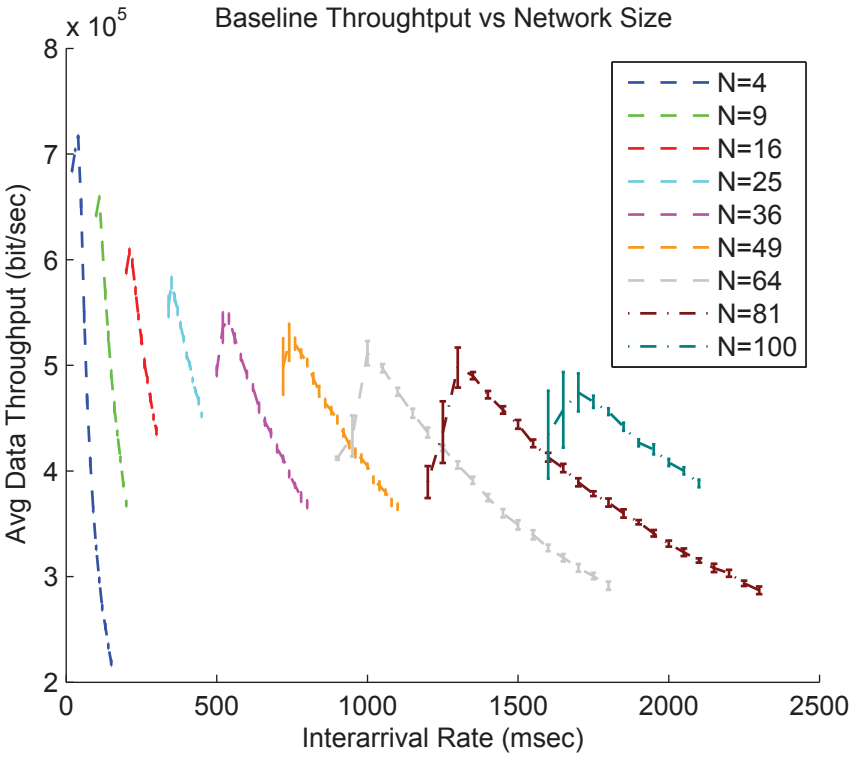


Figure 4.6: Baseline Network Throughput

4.2.2 Network throughput.

Network throughput is a measure of the amount of data that can be transferred through a network. Of all the metrics collected as part of this experiment, network

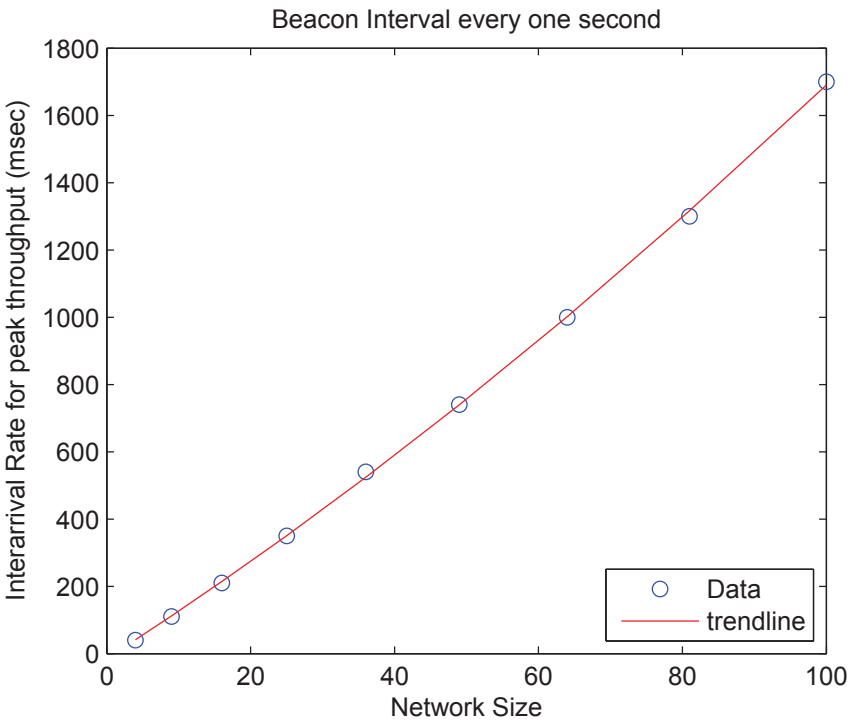


Figure 4.7: Correlation of Network Size and Interarrival Rate for Peak Throughput

throughput provided the most stable and useful properties for analysis. This section examines the throughput achieved by each protocol, and how REM exchanges affect performance compared to the baseline network.

4.2.2.1 Polling and Time-Division Protocols.

Both the Polling protocol and Time-Division protocol performed similarly in the simulations. The two protocols required each node in the network to broadcast their local REM during each exchange. The Polling protocol also included the need for a request packet (poll) to be sent to each node causing additional overhead.

In examining throughput, the overhead costs increased as the beacon interval become more frequent. Figure 4.8 shows the throughput changes for a 16 node network as a result of varying the exchange interval. Other network sizes displayed similar results of decreased throughput, but at different magnitudes of change. This behavior is to be expected since the more exchanges that occurred allowed less normal traffic data to be

transmitted. As a bound, consider that the beacon interval must be greater than the time required for an exchange to occur. For the simulations, any beacon interval more frequent than this bound caused the simulation to crash. The examination of the 16 node network was chosen over the other network sizes because of the clear separation of the data lines. Figure 4.8 as a representative sampling of the other network sizes also showed a non-linear relationship to the overhead costs as the interval changed. A linear relationship would display a constant change in throughput for the interval rates tested. As described in Table 3.3, the simulation of each network size used a unique set of exchange intervals for testing. Each network size contained a beacon interval of one second to allow a direct comparison of the REM protocols.

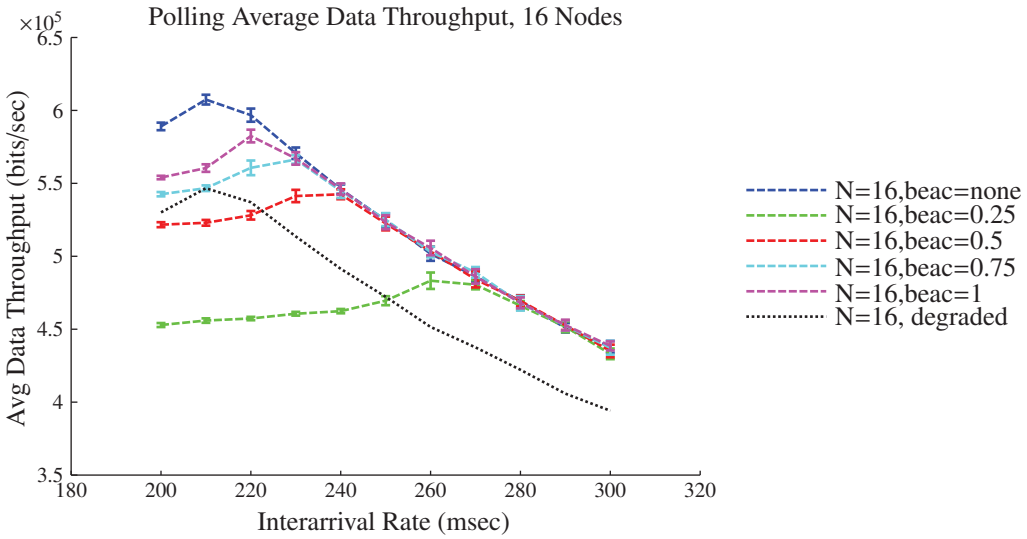


Figure 4.8: Polling Protocol - Throughput for a 16 node network

Taking the peak throughput for each network size at a one second beacon interval produced Figure 4.9. This figure displays a comparison of the peak throughput for the Polling protocol and the Time-Division protocol against the baseline. As shown in the figure, the use of the Time-Division protocol produced average throughput values higher than the Polling protocol. The addition of the polling packets explains the difference in the

throughput levels. The deviation from the baseline performance increased as the number of nodes in the network increased. Figure 4.10 shows this deviation as a percentage of the baseline throughput. Note that both protocols function worse as the network size grew.

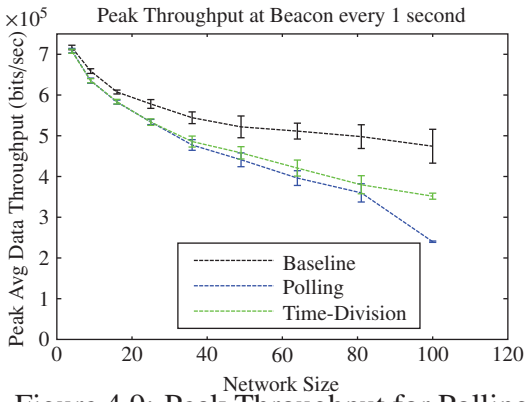


Figure 4.9: Peak Throughput for Polling and Time-Division protocols

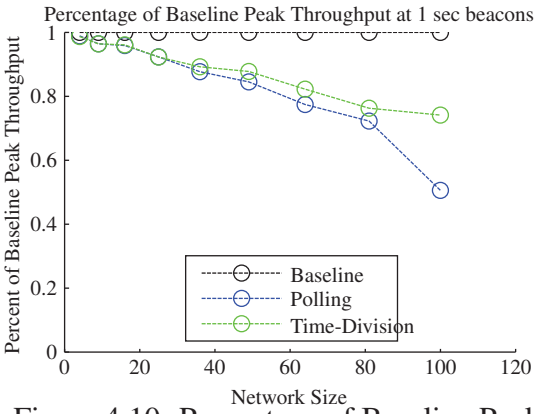
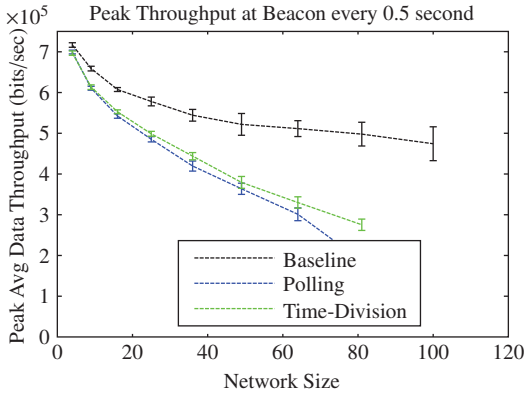
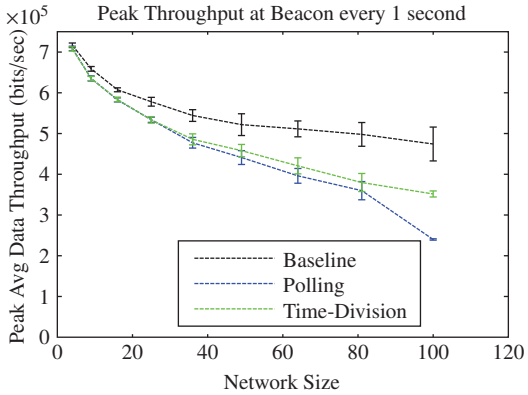


Figure 4.10: Percentage of Baseline Peak Throughput for Polling and Time-Division protocols

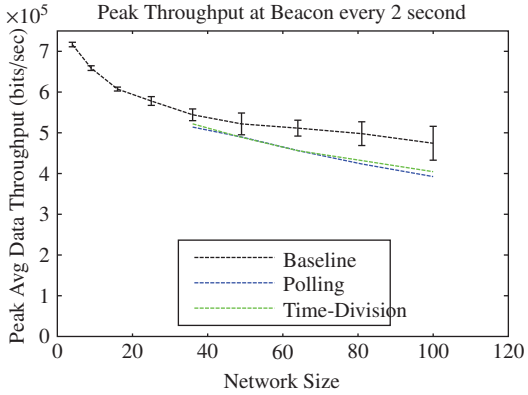
A side-by-side comparison of three different beacon rates reinforced that deviation from the baseline is minimized as the REM exchange interval increases. Figure 4.11 shows the deviations, and how both the Polling protocol and Time-Division protocol performed similarly each situation.



Beacon at 0.5 second interval



Beacon at 1 second interval



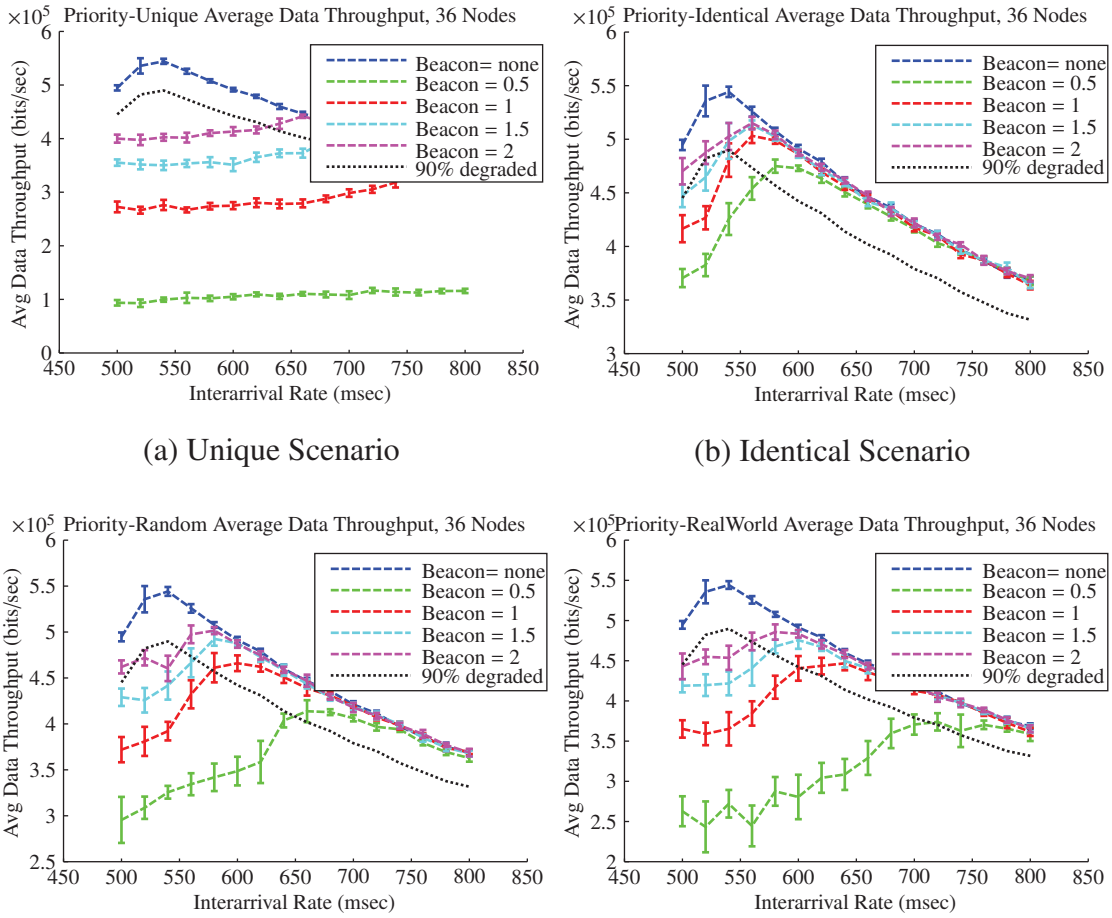
Beacon at 2 second interval

Figure 4.11: Beacon Interval Throughput Comparison

4.2.2.2 Priority Protocol.

The simulations testing of the Priority protocol required testing four scenarios. The other protocols only required a single scenario since every node was required to transmit its individual REM during each exchange. The Priority protocol, however, functions based on the similarity in the REMs of the network nodes. The four scenarios created upper and lower boundaries of operation as well as provided two examples of possible operational conditions. The Unique scenario was the lower bound scenario which represented a situation when every node contained at least one unique channel restriction resulting the most network activity. The Identical scenario was the upper bound scenario where all nodes contained the same local REM requiring only minimal network activity. Lastly, the Random and Real World scenarios represented two conditions where parts of the RF environment are shared among nodes. As an representative example of the data collected in simulation, Figure 4.12 displays the network throughput results for each scenario on a 36 node network. These graphics are presented to show the general differences produced by the scenarios. Appendix A contains complete data charts for each network size tested. The examination of the 36 node network was chosen over the other network sizes because of the clear separation of the data lines, and to allow a direct comparison against the data presented for the other protocols.

As with the Polling and Time-Division protocols, results for the Priority protocol in each scenario explored the effects of the interval between beacons. Comparing the peak average throughput of each scenario allowed for the expected operational range to be seen. Figure 4.13 expresses this comparison at a beacon interval of one second. The figure shows that the Unique scenario (lower bound) diverged from the baseline as the size of the network increased. The Identical scenario (upper bound) converged to the baseline throughput value as the network size increased. A major constraint in throughput arose as more collisions occurred. For larger networks within the simulation, each node's constant



(a) Unique Scenario

(b) Identical Scenario

(c) Random Scenario

(d) Real World Scenario

Figure 4.12: Priority Protocol - Throughput for a 36 node network

and equal amount of traffic drove the collision rate. The amount of normal data traffic that was sent during the REM exchange time was minimized due to the collisions, and this explains the convergence of the the baseline and Unique scenario values. All simulations in these experiments required each node in the network to share an equal traffic load. The Random and Real World scenarios performed as expected. Each produced statistically similar values for networks of small size, and tracked closer to the upper bound than the lower bound. To better understand the amount of nodal interaction occurring within the scenarios, the simulation collected additional information 100 node network size simulations to determine the average number of packets (overhead) successfully

transmitted during a REM exchange. This value provided awareness of the REM similarity that each scenario produced. The simulations for the Random scenario required an average of 6.9 packets with a standard deviation of 0.34 packets. The simulations for the Real World scenario required an average of 12.05 packets with a standard deviation of 2.24 packets. If collected, the Unique scenario would have produced a value of 100 packets, and the Identical scenario would have produced a value of zero packets. Together these values show strong correlations to the loss in average throughput from the baseline.

Viewed as percentage of the baseline, shown in Figure 4.14, its clear that the Random and Real World scenarios display a convergence to the baseline as the network size increased. At a network size of 100 nodes, the peak average throughput of both scenarios approach or exceed the 90% functionality threshold. The assumption that every node was within communication range during simulation limited the number of packets required in each REM exchange. However, since the design of the Real World scenario intended to cover many of the possible RF environments, this gives confidence that the number of exchanges in other real-world scenarios would also function with a limited number of exchanges being required.

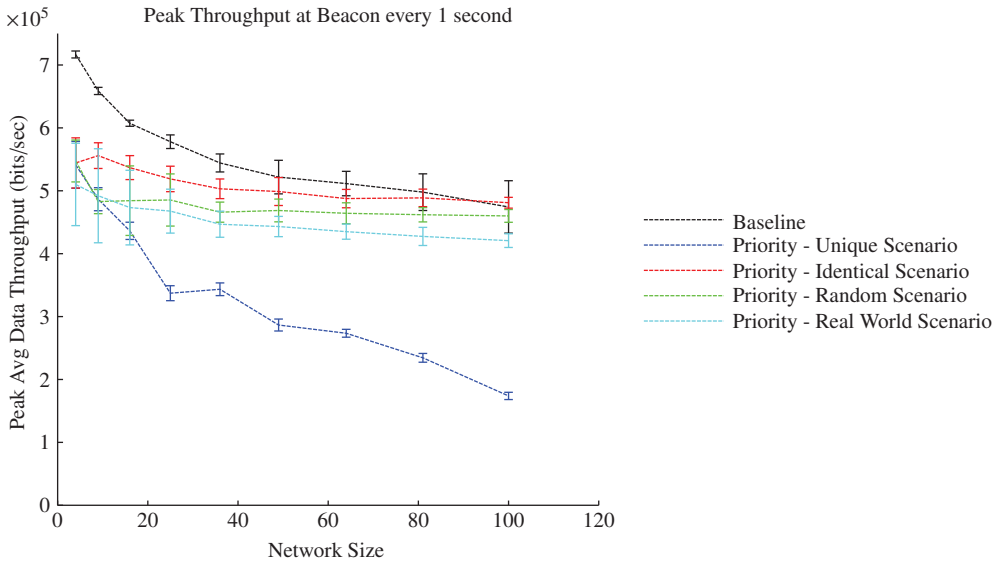


Figure 4.13: Priority Protocol - Peak Network Throughput (1 beacon/sec)

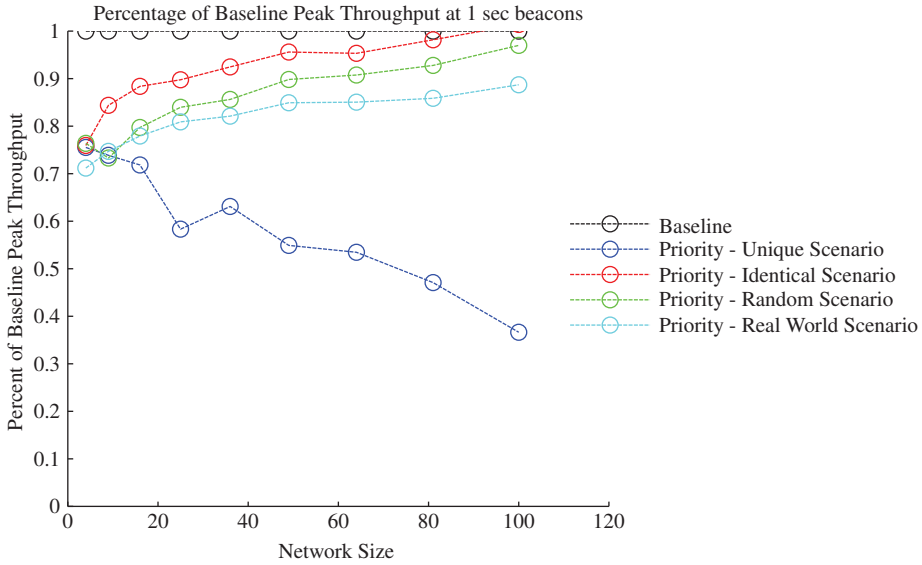


Figure 4.14: Priority Protocol - Percentage of Baseline Peak Throughput (1 beacon/sec)

4.2.3 Data dropped due to buffer overflow.

In the OPNET model, each node generated normal data traffic for the network to process. The amount of data created by each node was defined by the size of each packet and the interarrival rate of the node's packet generator. As shown in Table 3.2, each node consisted of a transmission buffer capable of holding $256K$ bits of normal (non-control) data. The buffer fills as packet generation exceeds a nodes ability to transmit the data. If a the data buffer is at max capacity, and if a new data packet arrives for the node, that new packet is discarded. The amount of data discarded was tracked by the data dropped due to buffer overflow metric.

For any REM exchange interval, the buffer drop rate maintained near zero packet loss until the critical packet generation rate was reached. This critical point indicated that maximum supported workload that the network could successfully process without loss occurring. Additionally, examining this point of first overflow revealed a correlation in the data workload to the occurrence of the peak throughput.

Figure 4.15 and Figure 4.16 display an example of this correlation. These figures show an 81 node network operating with the Time-Division protocol. The figures,

showing four different beacon intervals and the baseline, demonstrate that the occurrence of peak throughput and first buffer overflow correspond to the same network workload. This correlation is found through the simulation results regardless of network size or protocol. When more traffic is generated than the buffer can hold, the result is for the node always to have data available for transmission.

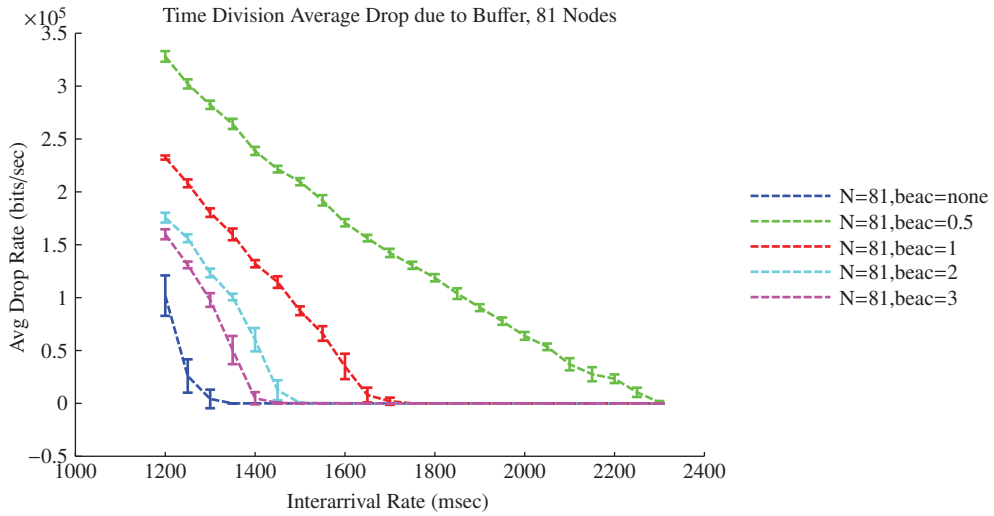


Figure 4.15: Data Drop due to Buffer Overflow, Time-Division Protocol, 81 nodes

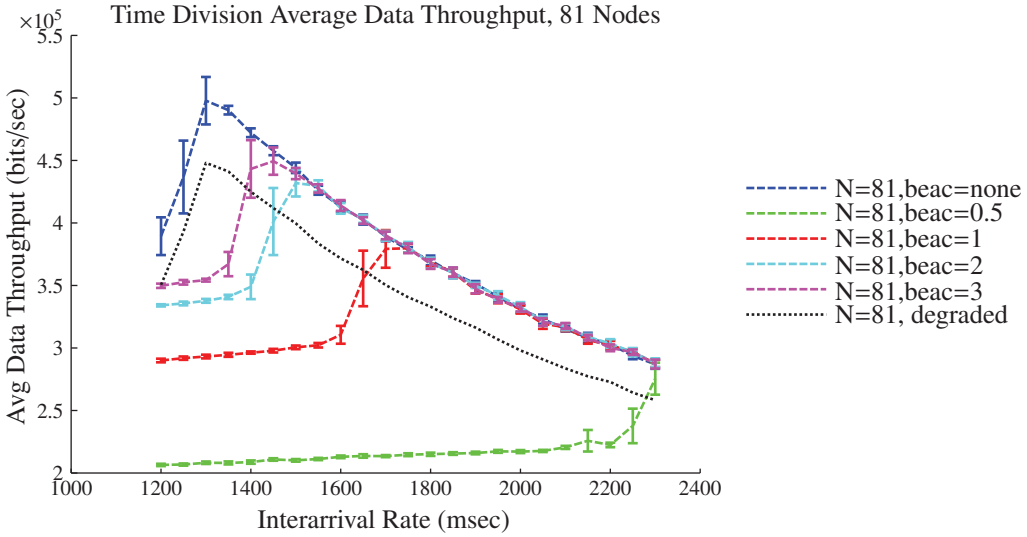


Figure 4.16: Average Throughput, Time-Division Protocol, 81 nodes

4.2.4 Data dropped due to exceeding the retry limit.

The data dropped due to the retry limit metric produced inadequate results for a comparison analysis. For most simulations, the metric reported values of zero. Investigation into the model's code showed a possible reason for these results. In the program code of the model, a normal data packet queued for re-transmission was dropped if interrupted by a REM exchange. No metric monitored this dropped packet, and it was lost in the system. The program assumed that the packet was properly handled resulting in a reset of the retry counter. A relatively short REM exchange interval therefore restricted a packet from reaching the maximum retry limit. Only packets reaching the limit were counted as part of the metric. This problem is in the implementation of the model and does not reflect the performance of the protocols. However, this zero value problem eliminates the dropped data due to exceeding the retry limit metric, and is omitted from further analysis of the results.

4.2.5 Retry attempts.

Attempting to track total retry attempts as a metric also proved problematic due to the model code issue. For the retry attempts metric, the Polling and Time-Division protocol displayed the most consistent results. Figure 4.17 illustrates the results of this metric for the 81 node network simulation using the Time-Division protocol. This figure verifies two expected characteristics for retry attempts. First, the maximum average retry attempts should be similar among REM update intervals with the maximum value as a function of network size. Second, as the exchange interval decreases, the workload required to reach the maximum average attempts should also decrease. Unlike the Polling and Time-Division protocol simulations, the Priority protocol produced results that were inconsistent with the expectations of a network's performance in terms of delay. Only a small amount of retry attempts were recorded when utilizing the Priority protocol. This was most likely due to the same model code as described above. Due to the inconsistency

in the reporting of the metric among protocols, the retry attempt metric was not investigated further in this analysis.

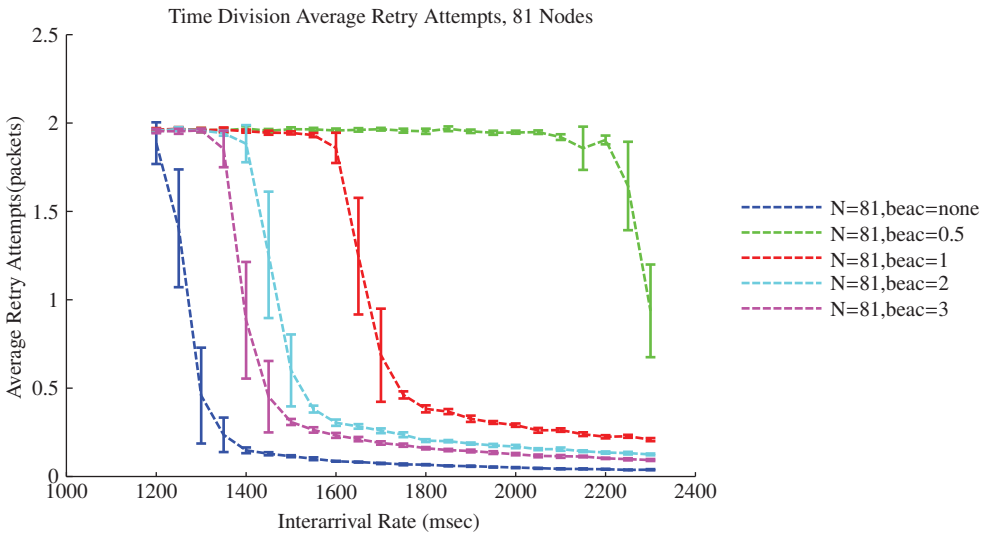


Figure 4.17: Average Retry Attempts, Time-Division Protocol, 81 nodes

4.2.6 Packet delay.

The delay metric collected the average amount of time to deliver a successful packet; however, it did not track the number of packets that were successfully delivered. The REM exchange interval and the protocol itself limit the total amount of traffic to successfully be delivered as seen by the network throughput results. If the system successfully transmitted only a single packet, then that one packet's delay would be used in defining the metric value. Even with this limitation in data collection, a useful insight can be gained. Figure 4.18 corresponds the same 81 node network described in the Section 4.2.3. This figure displays the alarming amount of delay that occurs in high workload network conditions. The delay quickly exceeds any acceptable level of delay at the same workload conditions that the buffer began to overflow. The coarseness of available data points do not allow for analysis of the max delay before the precise workload that causes the delay to rise; however, results before this critical workload seem to be similar to the results of the

baseline network. As such, the performance of the network is driven by the beacon interval, and the amount of time needed to perform a REM exchange.

The Polling and Time-Division protocols displayed similar similar delay growth for all network sizes. However, the Polling protocol required less of a workload to begin increases in delay. The collection of the packet delay metric for the Priority protocol resulted in inconclusive data which cannot be compared to the other protocols. Specifically, the results suggested that almost no delay was recorded among packets which indicates a problem in the collection methods. Packet delay may be a limiting performance issue when Quality-of-Service is required for network operation, but the results of this research do not provide the resolution needed to analyze such performance requirements.

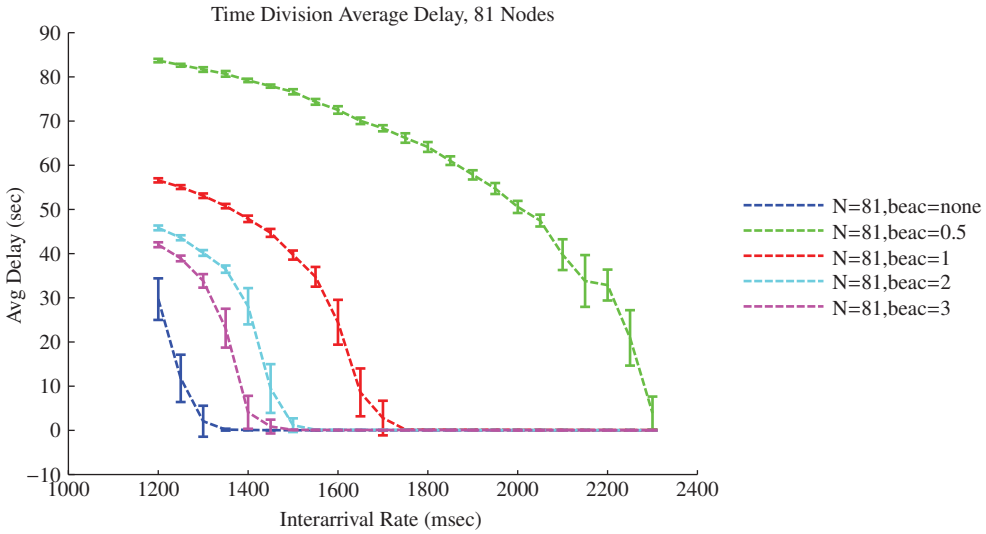


Figure 4.18: Average Retry Attempts, Time-Division Protocol, 81 nodes

4.2.7 Further Testing.

As stated above, several of the metrics could not be analyzed due to an error in packet handling within the program code of the model. This code error was identified and corrected. Data was then collected on this corrected model using the same methodology as described in Section 3.2. The analysis of this data proved to be statistically inconclusive

due to the large variations of the confidence intervals. An example of variations in the results are shown in Figure 4.19. The figure displays the peak throughput for the Priority protocol under each of the four scenarios. As shown, the confidence intervals constructed using 15 iterations of each simulation setup provides no statistical differences for conclusions to be drawn. To properly analyze this updated model additional iterations are required to reduce the 95% confidence intervals; however, due to time limitations these iterations will not performed as part of this research effort, but should be considered in any future work.

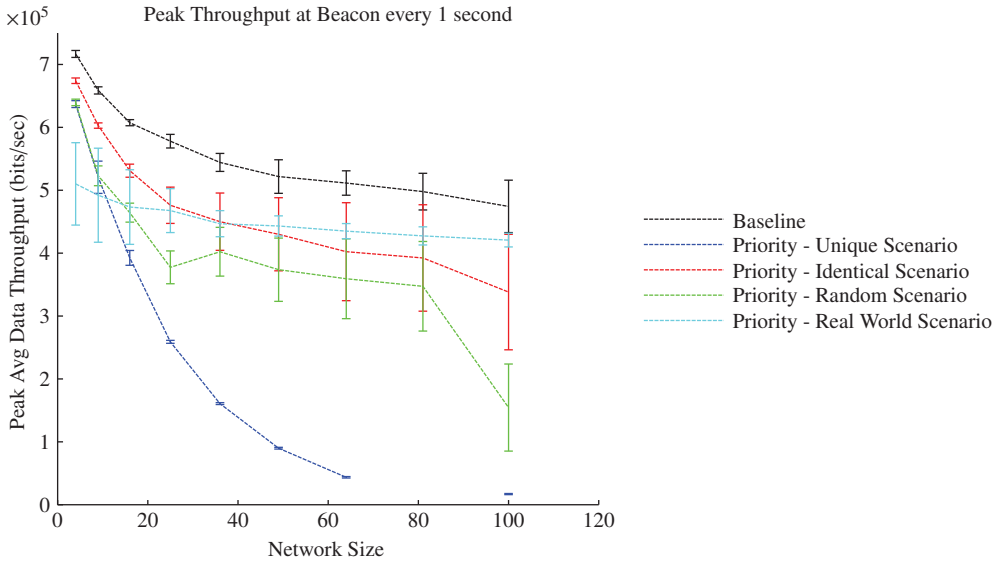


Figure 4.19: Peak Throughput for Priority protocol (updated results)

V. Conclusions

This chapter summarizes the goals and conclusions of this research. Section 5.1 presents the goals and results of the studies. Section 5.2 outlines the significance of this research. Lastly, Section 5.3 provides suggestions for future research into cognitive radio.

5.1 Research Goals Acheived

The first goal of this research was to demonstrate that Gold's algorithm allows for successful rendezvous on an FHSS radio network without dwell time or common control channel requirements. Development and demonstration of the FPGA based algorithm, as described in Section 4.1, showed successful rendezvous with Gold's Algorithm. The design proved that detection of a signal on a communication channel, regardless of dwell time or specific channel, allows for rendezvous to occur. The system also includes limitations in terms of operational applicability. The algorithm itself cannot handle false positives within signal detection. A false positive requires the algorithm to reset since a solution will not be found. This results in a possible jamming condition. Any reduction in the number of detections required by the algorithm greatly increases its rate of success.

The second goal was to analytically show that the above algorithm is capable of operating within the DSA paradigm. Section 4.1.4 discussed the advantages and disadvantages of methods to interpret a hopping sequence. The analysis of the skip, hold, and modulus methods showed that Gold's algorithm performs to the same limitations as the COD method of rendezvous for FHSS radio systems.

The third goal of this research was to present system hardware requirements for an FPGA implementation of Gold's algorithm. Research into Gold's algorithm as shown in Section 4.1 involved the development, testing, and analysis of an FPGA implementation of the algorithm at different sequence period lengths (LFSR bit length) and number of

communication channels (tap bits). The experiment revealed that only a general trend showing that increasing either variable causes a decrease in maximum operational clock frequency. This clock limitation resulted from significant routing delay more than delay due to logic. The resources required to build each system increase as system complexity increased. Of the two main reportable FPGA resources, registers and LUTs, a general equation was determined to predict the number of registers required. The LUT usage of the system did not conform to a predictable pattern with reliable results. The hypothesis was disproved in that all resources required are not directly calculable. These results are specific to synthesis using Xilinx ISE 13.2 and the XST synthesis tool targeted to the Virtex-5 FPGA.

The last goal of this research was to evaluate the performance expectations of three communication methods for exchanging radio environment data on a multi-nodal radio network via simulation. The OPNET simulation experiment revealed that both the Polling and Time-Division protocols required less overhead costs to average network throughput at small sized networks. These protocols maintained above 90% maximum throughput upto a network size of approximately 35 nodes when the a REM exchange occurred every 1 second. As network size increases, the Priority protocol performed the best method of the three tested under near real-world scenarios. The Real World scenario used in testing never met the 90% threshold; however, extrapolation of trends show that any increase in network size above the levels tested should meet the threshold. The experiment also showed that the addition of nodes to a static network where all nodes are within communication range did not increase overhead costs in larger networks when using the Priority protocol. These results match that of the hypothesis. Each REM exchange protocol displayed points of advantage and disadvantage as network traffic and network size increase. No protocol performed best over all tests, and therefore the selection of the exchange protocol becomes a factor within a cognitive system.

5.2 Research Contributions

This research provides two distinct contributions to the study of DSA, and therefore the field of CR. First, this research establishes Gold's algorithm as a valid rendezvous method for Frequency Hopping DSA networks. Compared to traditional methods such as COD, Gold's algorithm reduces rendezvous time when higher hop rates are required or significant time has elapsed since the key was issued. Additionally, this research offers the first FPGA hardware implementation of Gold's algorithm providing timing and resource requirements for radio designers.

Second, this research demonstrates the impact to network performance for three exchange protocols when radio channel availability data must be exchanged as part of DSA operations. Examination of metrics such as data throughput, delay, and data drop rates provide an estimation of network performance with varying network size under different network non-control traffic workloads. Furthermore, a simulation test scenario to estimate real-world conditions is presented due to the lack of actual real-world representative test sets.

Two OPNET nodal models were developed to compare the exchange protocols. Utilization of these models provide future researchers with advanced capabilities not currently available. Therefore, these models can be considered minor contributions. The first model expands the features of the IEEE 802.11b OPNET wireless model to incorporate the transmission technique of FHSS. OPNET's model provides frequency hopping as a function option, but the feature was never implemented in the model. The improved model is currently restricted to slow-hopping operations. The second model modifies the PCF of IEEE 802.11b within the MAC process model. This modification allows PCF operation without the need of a dedicated access point, and leverages the PCF operation for transfer of radio environment data only.

5.3 Future Work

One of the most difficult problems developing this research was the lack of environmental RF data to be used for experimentation. No open source data sets were discovered to create a real-world operation simulation, and therefore a representative scenario had to be created. One suggestion for future work includes the collection and analysis of the RF environment to create a test set. This work would provide researchers with realistic conditions for testing. It also could lead to studies in environment prediction algorithms. Secondly, the simulations performed within this research were limited to stationary nodal models within communication range. Future research into the affects that mobility could cause in the exchange process is needed. Additionally, the inclusion of existing routing protocols and application layer simulation would further the study of reliability in a cognitive network.

Although network simulations provide needed insight into wireless network performance, hardware implementation and testing is the ultimate goal. The custom IP core for Gold's algorithm needs to be transitioned to an RF capable platform for further testing. The exchange protocols researched in this paper may also be implemented on a development board for further testing. AFIT is dedicated to developing a physical cognitive radio capable of dynamically selecting rendezvous methods, routing protocols, and transmission waveforms to optimize performance. Each of these areas should be explored and possibly implemented.

Continuing research into cognitive radio allows for better utilization of a limited resource. Radios equipped with the ability to actively select the best operating parameters provides flexible and reliable communication through a increasingly chaotic medium. For military use, a cognitive system could make the difference between a successful mission and complete failure. Future research by AFIT will assist in advancing our communication technology for a stronger United States of America.

Appendix: Experimental Results

A.1 FPGA implementation Results

		Device utilization summary							HDL Synthesis Report							
		Slice Utilization		Slice Distribution					Macro Statistics							
Taps	Width	# Slice Registers	# Slice LUTS	# of LUT FF pairs	fully used	Control sets	IOs	BUFGs	RAMS	Adder/Sub	Counters	Registers	Latches	Comparators	Multiplexers	XORs
4	8	500	801	969	332	36	52	4	1	18	5	220	9	16	3	81
4	12	865	1404	1748	521	44	56	4	1	23	5	404	13	16	3	169
4	16	1363	1962	2546	779	52	60	4	1	26	5	652	17	16	3	289
4	20	1983	2706	3595	1094	60	64	4	1	31	5	964	21	16	3	441
4	24	2731	3652	4905	1478	68	68	4	1	35	5	1340	25	16	3	625
4	28	3607	4713	6398	1922	76	72	4	1	39	5	1780	29	16	3	841
4	32	4617	5792	7977	2432	84	76	4	1	42	5	2284	33	16	3	1089
4	36	5749	7198	9942	3005	92	80	4	1	47	5	2852	37	16	3	1369
5	8	521	877	1044	354	37	53	4	1	18	5	236	9	16	3	81
5	12	886	1499	1842	543	45	57	4	1	23	5	420	13	16	3	169
5	16	1384	2051	2635	800	53	61	4	1	26	5	668	17	16	3	289
5	20	2005	2713	362	1116	61	65	4	1	31	5	980	21	16	3	441
5	24	2753	3718	4973	1498	69	69	4	1	35	5	1356	25	16	3	625
5	28	3628	4745	6432	1941	77	73	4	1	39	5	1796	29	16	3	841
5	32	4638	5942	8128	2452	85	77	4	1	42	5	2300	33	16	3	1089
5	36	5770	7277	10026	3021	93	81	4	1	47	5	2868	37	16	3	1369
6	8	557	1059	1220	396	38	54	4	1	18	5	268	9	16	3	81
6	12	923	1498	1841	580	46	58	4	1	23	5	452	13	16	3	169
6	16	1421	2131	2716	836	54	62	4	1	26	5	700	17	16	3	289
6	20	2041	2838	3731	1148	62	66	4	1	31	5	1012	21	16	3	441
6	24	2789	3822	5077	1534	70	40	4	1	35	5	1388	25	16	3	625
6	28	3665	4845	6530	1980	78	74	4	1	39	5	1828	29	16	3	841
6	32	4675	5980	8169	2486	86	78	4	1	42	5	2332	33	16	3	1089
6	36	5807	7453	201	3059	94	82	4	1	47	5	2900	37	16	3	1369
7	8	627	1110	1278	459	39	55	4	1	18	5	332	9	17	3	81
7	12	992	1638	1983	647	47	59	4	1	23	5	516	13	17	3	169
7	16	1492	2240	2823	909	55	63	4	1	26	5	764	17	17	3	289
7	20	2210	2955	3845	1220	63	67	4	1	31	5	1076	21	17	3	441
7	24	2858	3910	5167	1601	71	71	4	1	35	5	1452	25	17	3	625
7	28	3734	4969	6662	2041	79	75	4	1	39	5	1892	29	17	3	841
7	32	4744	6263	8453	2554	87	7	4	1	42	5	2396	33	17	3	1089
7	36	5876	7647	10398	3125	95	83	4	1	47	5	2964	37	17	3	1369
8	8															
8	12	1126	1837	2182	781	48	60	4	1	23	5	644	13	17	3	169
8	16	1628	2416	3002	1042	56	64	4	1	26	5	892	17	17	3	289
8	20	2244	3146	4036	1354	64	68	4	1	31	5	1204	21	17	3	441
8	24	2992	4132	5388	1736	72	72	4	1	35	5	1580	25	17	3	625
8	28	3868	5197	6892	2173	80	76	4	1	39	5	2020	29	17	3	841
8	32	4878	6347	8534	2691	88	80	4	1	42	5	2524	33	17	3	1089
8	36	6010	8006	759	3257	96	84	4	1	47	5	3092	37	17	3	1369
9	8															
9	12	1387	2159	2502	1044	49	61	4	1	23	5	900	13	17	3	169
9	16	1885	2830	3412	1303	57	65	4	1	26	5	1148	17	17	3	289
9	20	2505	3578	4470	1613	65	69	4	1	31	5	1460	21	17	3	441
9	24	3253	4647	5906	1994	73	73	4	1	35	5	1836	25	17	3	625
9	28	4129	5672	7369	2432	81	77	4	1	39	5	2276	29	17	3	841
9	32	5139	6875	9065	2949	89	81	4	1	42	5	2780	33	17	3	1089
9	36	6271	8392	11143	3520	97	85	4	1	47	5	3348	37	17	3	1369
10	8															
10	12	1904	2762	3231	1435	50	62	4	1	23	5	1412	13	17	3	169
10	16	2402	3497	4116	1783	58	66	4	1	26	5	1660	17	17	3	289
10	20	3022	4211	5135	2098	66	70	4	1	31	5	1972	21	17	3	441
10	24	3770	5398	6732	2436	74	74	4	1	35	5	2348	25	17	3	625
10	28	4646	6491	8257	2880	82	78	4	1	39	5	2788	29	17	3	841
10	32	5656	7563	9819	3400	90	82	4	1	42	5	3292	33	17	3	1089
10	36	6788	9093	11902	3979	98	86	4	1	47	5	3860	37	17	3	1369

Figure A.1: FPGA Device Utilzation Summary and HDL Synthesis Report

		Advanced HDL Synthesis Report									Final Report					
		Macro Statistics									Cell Usage					
Taps	**Width**	FSM	RAMS	Adder/Sub	Counters	Registers	Latches	Comparators	Multiplexers	XORs	BELS	FF/Latch	RAMS	Clock Buf	IO buf	RAM type
4	8	1	1	18	5	546	9	16	2	81	1058	500	1	4	52	RAMB18
4	12	1	1	23	5	910	13	16	2	169	1666	865	1	4	56	RAMB18
4	16	1	1	26	5	1407	17	16	2	289	2284	1363	1	4	60	RAMB18
4	20	1	1	31	5	2027	21	16	2	441	3034	1986	1	4	64	RAMB36_EXP
4	24	1	1	35	5	2775	25	16	2	625	3958	2731	1	4	68	RAMB36_EXP
4	28	1	1	39	5	3651	29	16	2	841	5058	3607	1	4	72	RAMB36_EXP
4	32	1	1	42	5	4660	33	16	2	1089	6232	4617	1	4	76	RAMB36_EXP
4	36	1	1	47	5	5792	37	16	2	1369	7585	5749	1	4	80	RAMB36_EXP
5	8	1	1	18	5	566	9	16	2	81	1136	521	1	4	53	RAMB18
5	12	1	1	23	5	930	13	16	2	169	1761	886	1	4	57	RAMB18
5	16	1	1	26	5	1427	17	16	2	289	2389	1384	1	4	61	RAMB18
5	20	1	1	31	5	2047	21	16	2	441	3022	2005	1	4	65	RAMB36_EXP
5	24	1	1	35	5	2795	25	16	2	625	4033	2753	1	4	69	RAMB36_EXP
5	28	1	1	39	5	3671	29	16	2	841	5150	3628	1	4	73	RAMB36_EXP
5	32	1	1	42	5	4680	33	16	2	1089	6440	4638	1	4	77	RAMB36_EXP
5	36	1	1	47	5	5812	37	16	2	1369	7609	5770	1	4	81	RAMB36_EXP
6	8	1	1	18	5	602	9	16	2	81	1322	557	1	4	54	RAMB18
6	12	1	1	23	5	966	13	16	2	169	1779	923	1	4	58	RAMB18
6	16	1	1	26	5	1463	17	16	2	289	2501	1421	1	4	62	RAMB18
6	20	1	1	31	5	2083	21	16	2	441	3150	2041	1	4	66	RAMB36_EXP
6	24	1	1	35	5	2831	25	16	2	625	4147	2789	1	4	70	RAMB36_EXP
6	28	1	1	39	5	3707	29	16	2	841	5260	3665	1	4	74	RAMB36_EXP
6	32	1	1	42	5	4716	33	16	2	1089	6347	4675	1	4	78	RAMB36_EXP
6	36	1	1	47	5	5848	37	16	2	1369	7866	5807	1	4	82	RAMB36_EXP
7	8	1	1	18	5	670	9	17	2	81	1392	627	1	4	55	RAMB18
7	12	1	1	23	5	1034	13	17	2	169	1939	992	1	4	59	RAMB18
7	16	1	1	26	5	1531	17	17	2	289	2635	1492	1	4	63	RAMB18
7	20	1	1	31	5	2151	21	17	2	441	3299	2110	1	4	67	RAMB36_EXP
7	24	1	1	35	5	2899	25	17	2	625	4257	2858	1	4	71	RAMB36_EXP
7	28	1	1	39	5	3775	29	17	2	841	5368	3734	1	4	75	RAMB36_EXP
7	32	1	1	42	5	4784	33	17	2	1089	6771	4744	1	4	79	RAMB36_EXP
7	36	1	1	47	5	5916	37	17	2	1369	8066	5876	1	4	83	RAMB36_EXP
8	8															
8	12	1	1	23	5	1167	13	17	2	169	2164	1126	1	4	60	RAMB18
8	16	1	1	26	5	1664	17	17	2	289	2820	1628	1	4	64	RAMB18
8	20	1	1	31	5	2284	21	17	2	441	3501	2244	1	4	68	RAMB36_EXP
8	24	1	1	35	5	3032	25	17	2	625	4520	2992	1	4	72	RAMB36_EXP
8	28	1	1	39	5	3908	29	17	2	841	5617	3868	1	4	76	RAMB36_EXP
8	32	1	1	42	5	4917	33	17	2	1089	6762	4878	1	4	80	RAMB36_EXP
8	36	1	1	47	5	6049	37	17	2	1369	8411	6010	1	4	84	RAMB36_EXP
9	8															
9	12	1	1	23	5	1427	13	17	2	169	2540	1387	1	4	61	RAMB18
9	16	1	1	26	5	1924	17	17	2	289	3268	1885	1	4	65	RAMB18
9	20	1	1	31	5	2544	21	17	2	441	3982	2505	1	4	69	RAMB36_EXP
9	24	1	1	35	5	3292	25	17	2	625	5105	3253	1	4	73	RAMB36_EXP
9	28	1	1	39	5	4168	29	17	2	841	6164	4129	1	4	77	RAMB36_EXP
9	32	1	1	42	5	5177	33	17	2	1089	7365	5139	1	4	81	RAMB36_EXP
9	36	1	1	47	5	6309	37	17	2	1369	8850	6271	1	4	85	RAMB36_EXP
10	8															
10	12	1	1	23	5	1943	13	17	2	169	3216	1904	1	4	62	RAMB18
10	16	1	1	26	5	2440	17	17	2	289	4043	2402	1	4	66	RAMB18
10	20	1	1	31	5	3060	21	17	2	441	4742	3022	1	4	70	RAMB36_EXP
10	24	1	1	35	5	3808	25	17	2	625	5913	3770	1	4	74	RAMB36_EXP
10	28	1	1	39	5	4684	29	17	2	841	7093	4646	1	4	78	RAMB36_EXP
10	32	1	1	42	5	5693	33	17	2	1089	8137	5656	1	4	82	RAMB36_EXP
10	36	1	1	47	5	6825	37	17	2	1369	9640	6788	1	4	86	RAMB36_EXP

Figure A.2: FPGA Advanced HDL Synthesis Report and Final Report

A.2 Baseline (Pilot Study) Results

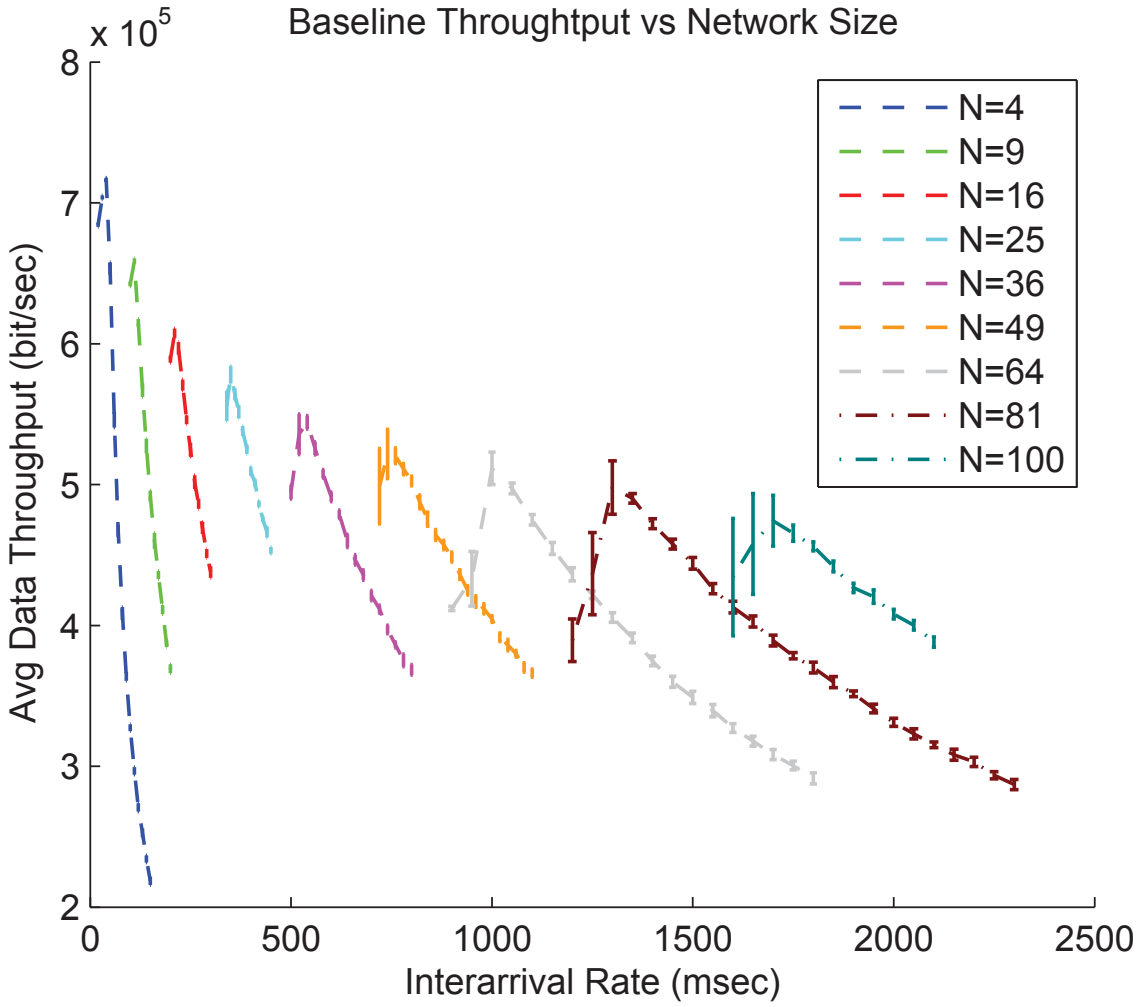


Figure A.3: Baseline Throughput

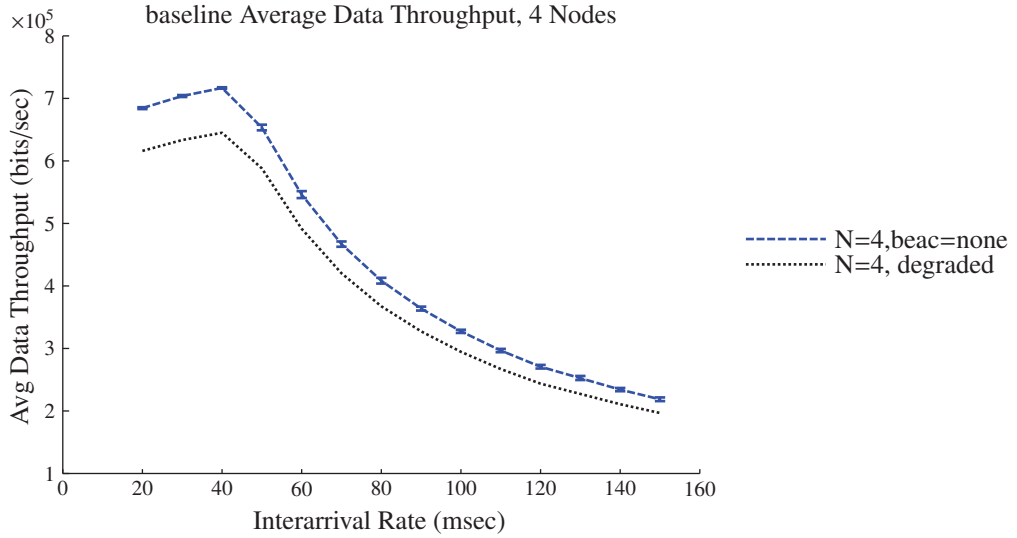


Figure A.4: N4 Data Throughput

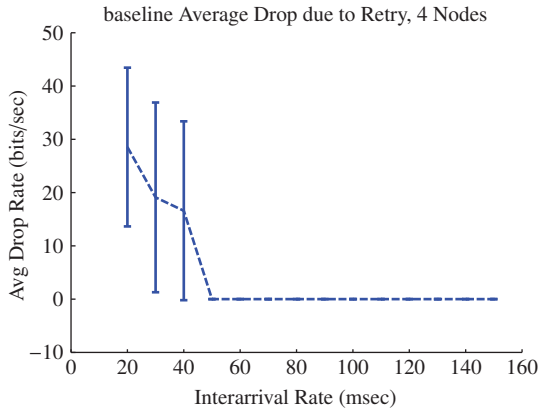


Figure A.5: N4 Retry Dropped Data

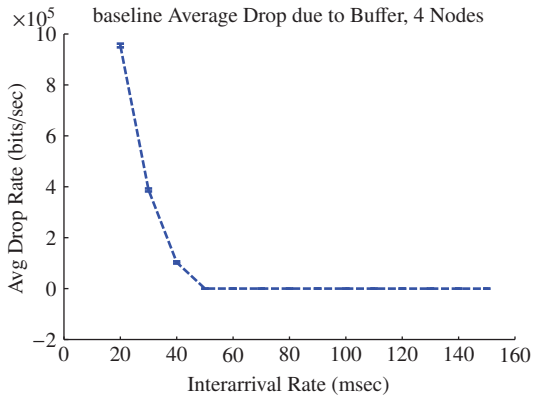


Figure A.6: N4 Buffer Dropped Data

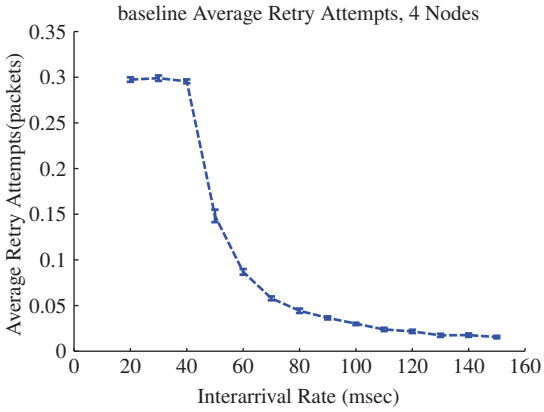


Figure A.7: N4 Retry Attempts

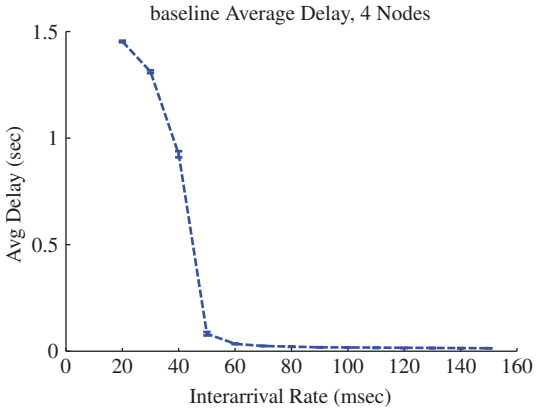


Figure A.8: N4 Packet Delay

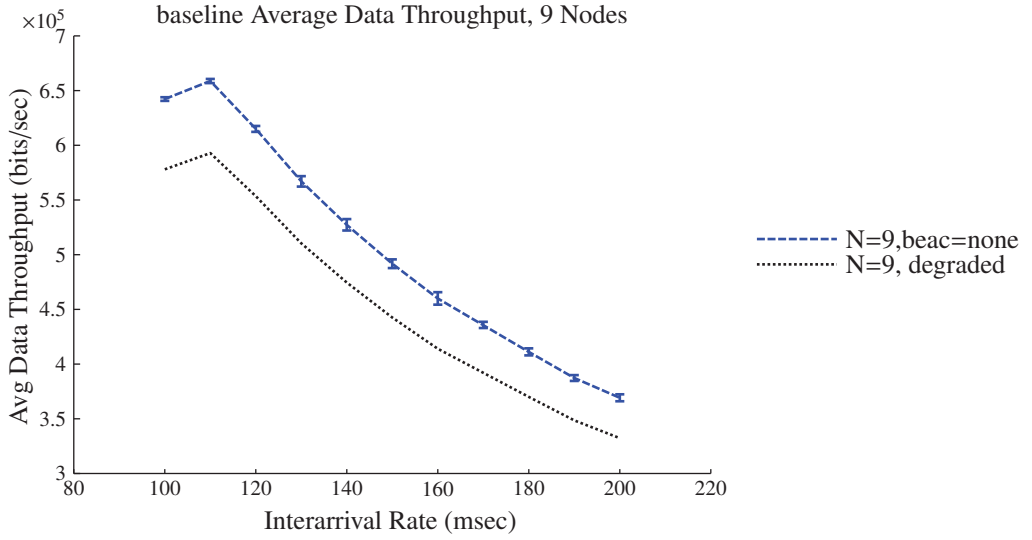


Figure A.9: N9 Data Throughput

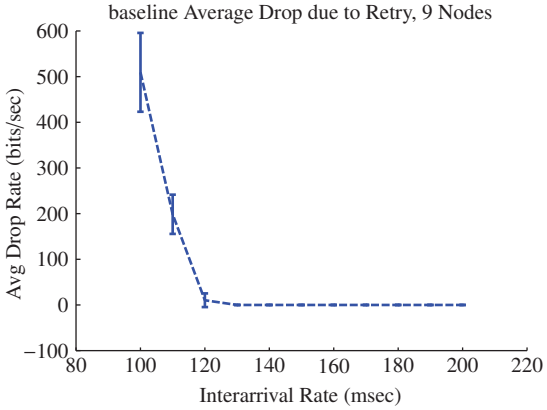


Figure A.10: N9 Retry Dropped Data

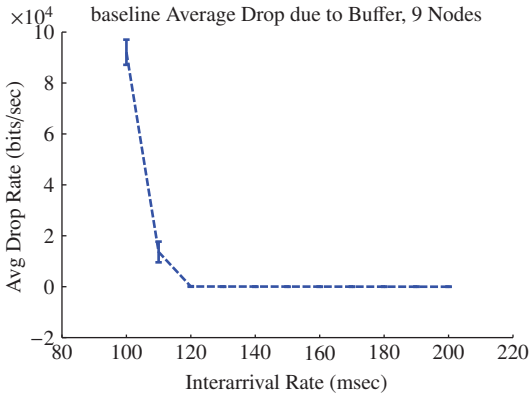


Figure A.11: N9 Buffer Dropped Data

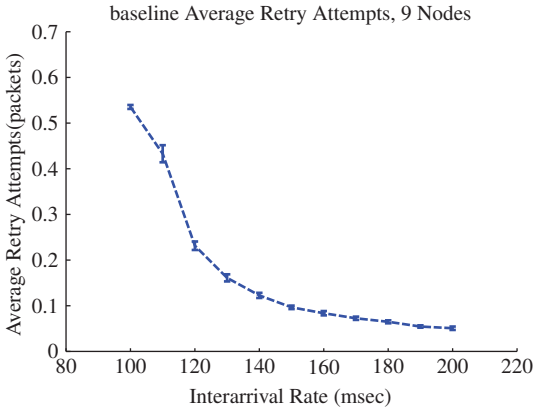


Figure A.12: N9 Retry Attempts

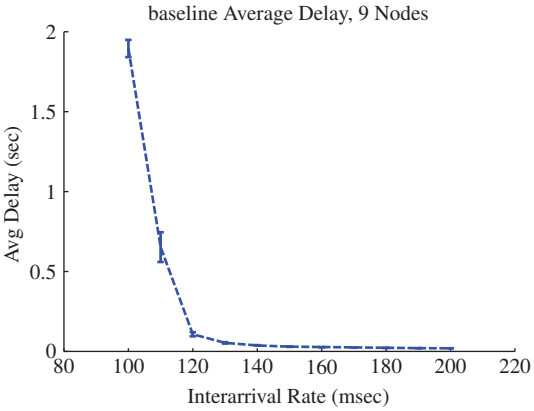


Figure A.13: N9 Packet Delay

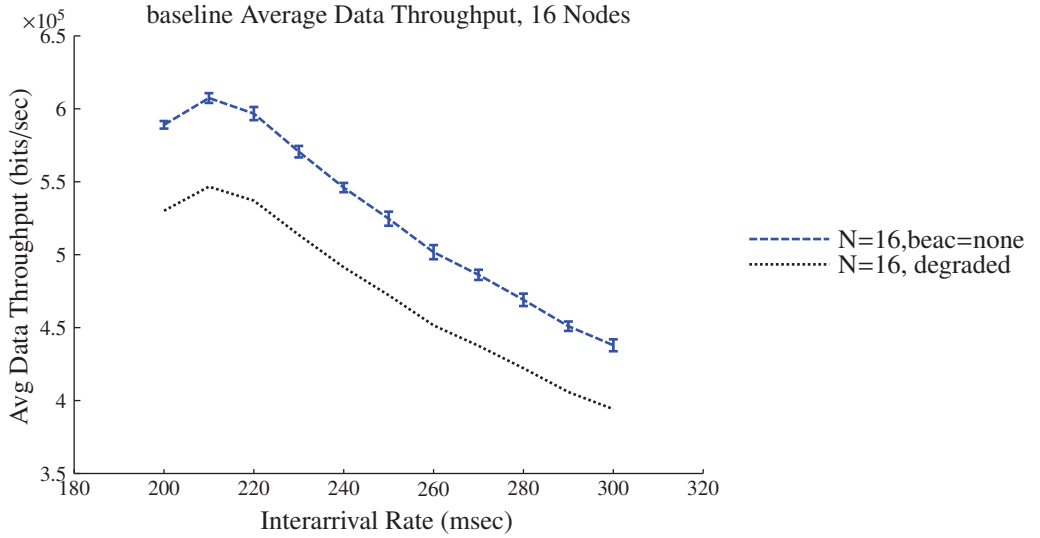


Figure A.14: N16 Data Throughput

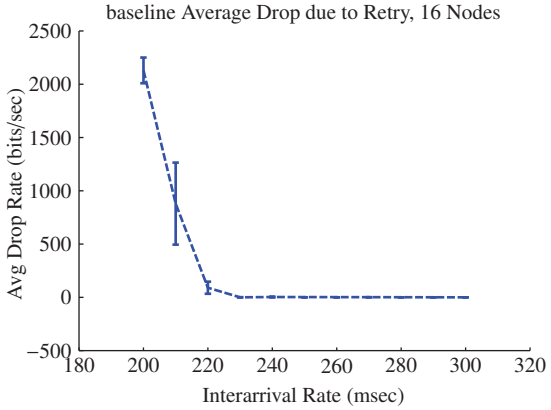


Figure A.15: N16 Retry Dropped Data

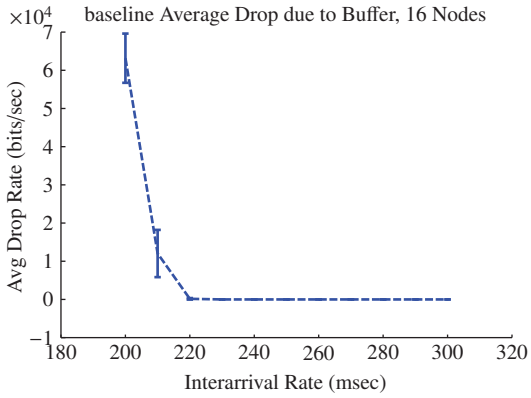


Figure A.16: N16 Buffer Dropped Data

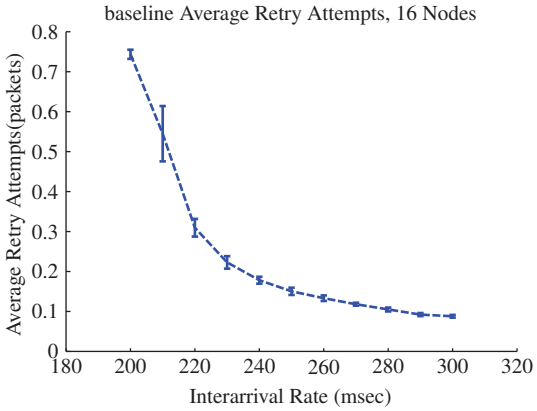


Figure A.17: N16 Retry Attempts

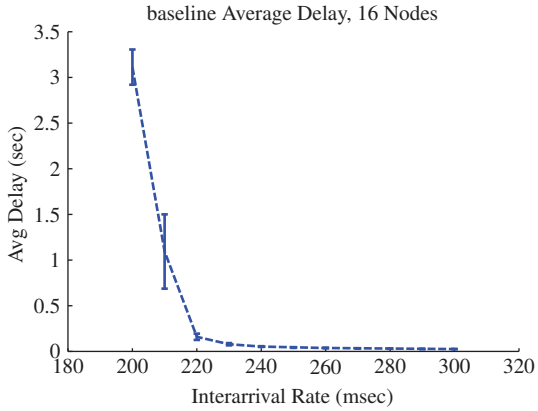


Figure A.18: N16 Packet Delay

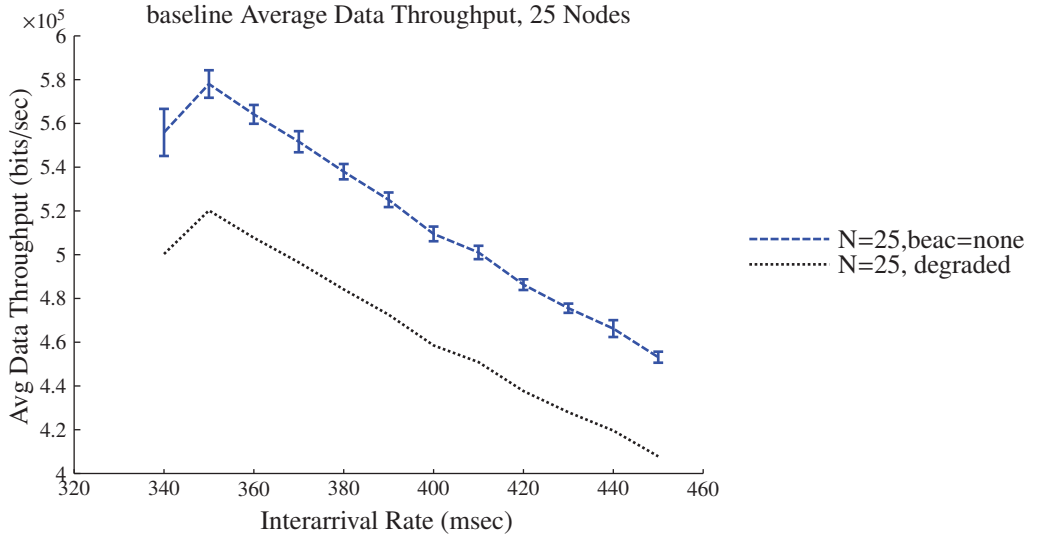


Figure A.19: N25 Data Throughput

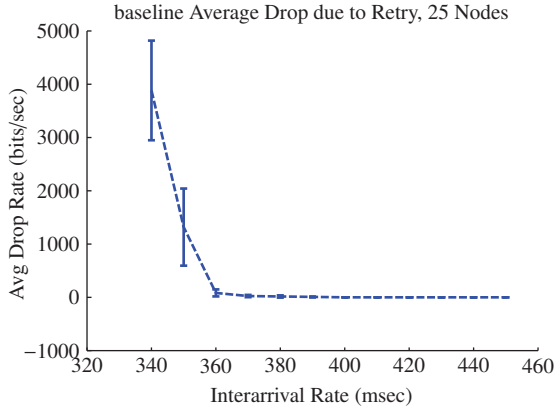


Figure A.20: N25 Retry Dropped Data

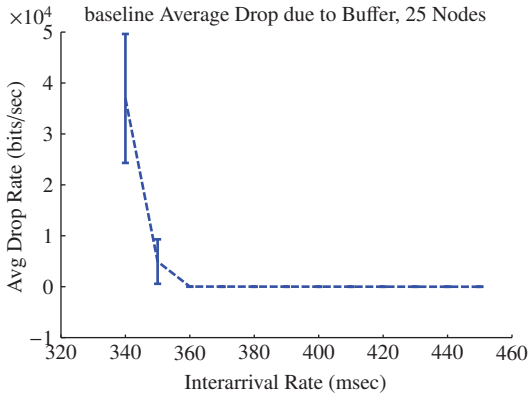


Figure A.21: N25 Buffer Dropped Data

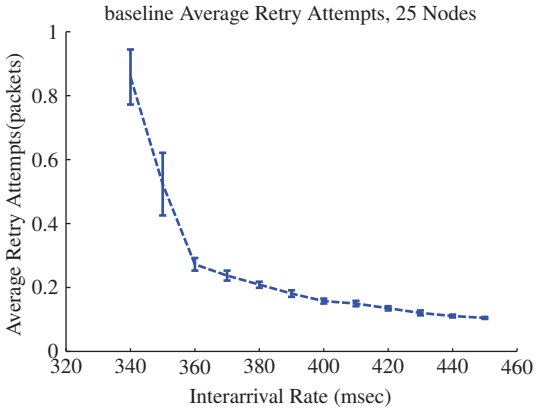


Figure A.22: N25 Retry Attempts

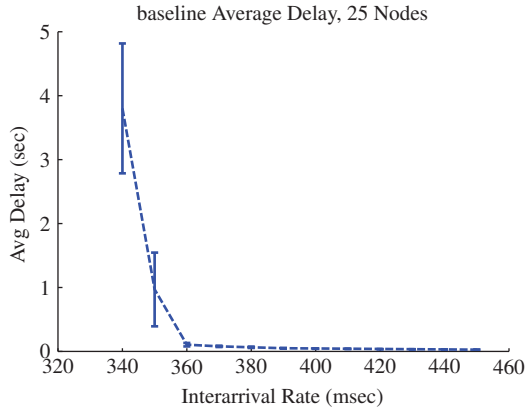


Figure A.23: N25 Packet Delay

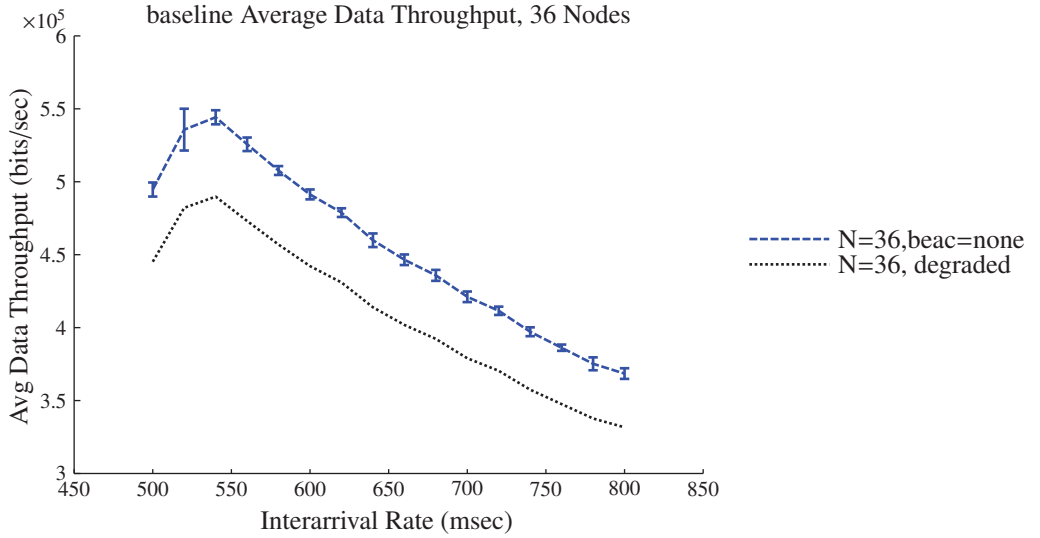


Figure A.24: N36 Data Throughput

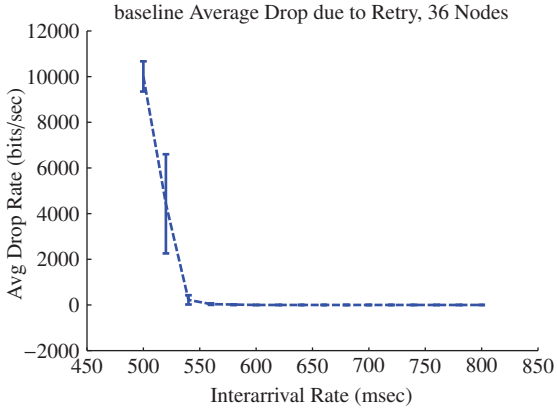


Figure A.25: N36 Retry Dropped Data

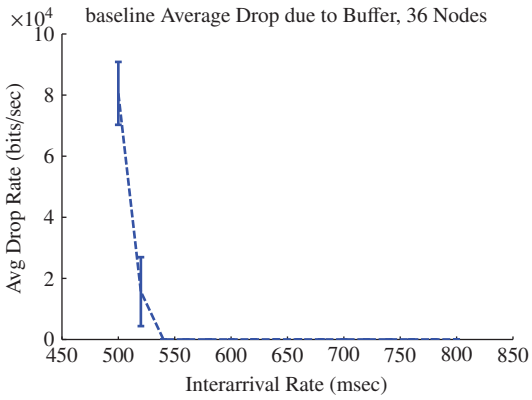


Figure A.26: N36 Buffer Dropped Data

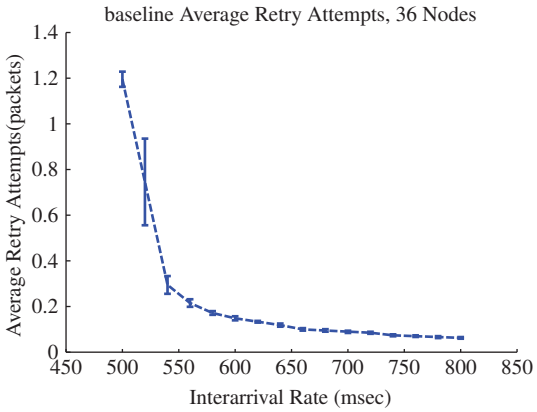


Figure A.27: N36 Retry Attempts

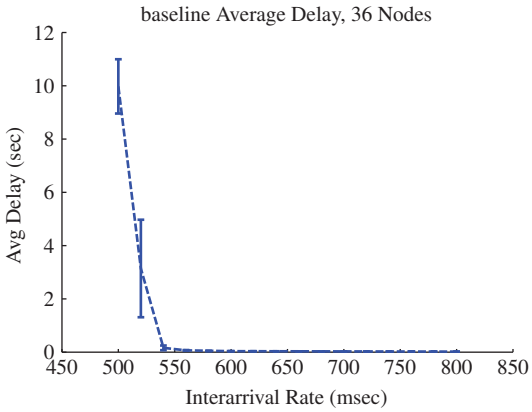


Figure A.28: N36 Packet Delay

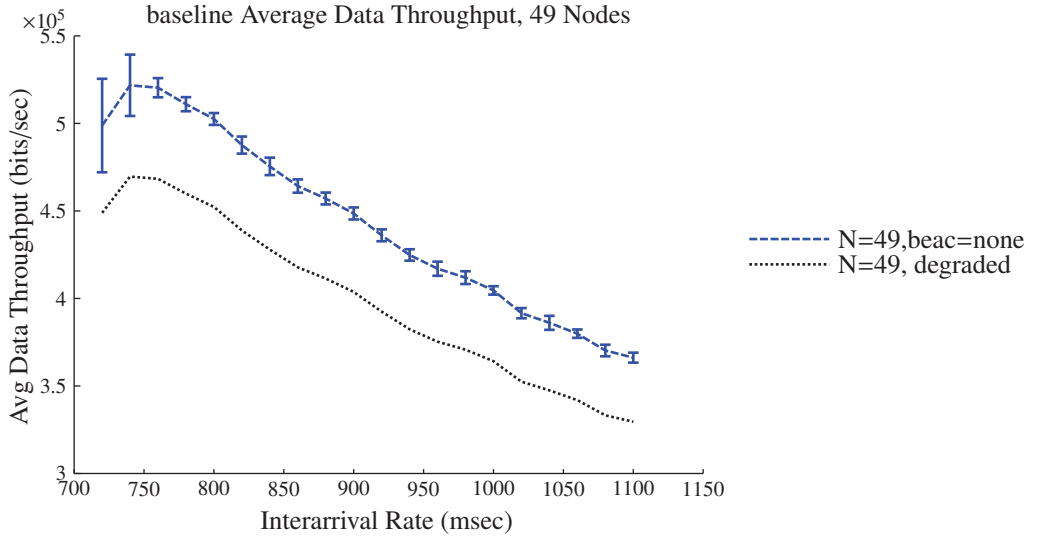


Figure A.29: N49 Data Throughput

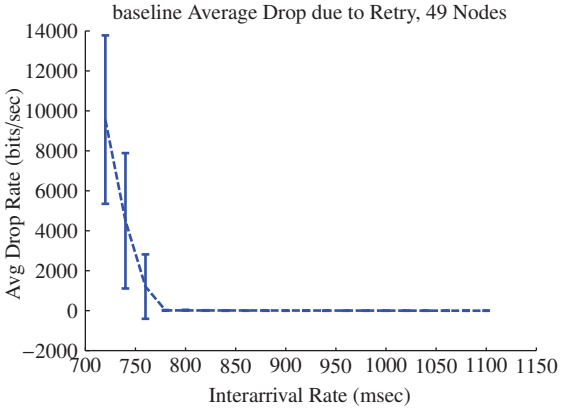


Figure A.30: N49 Retry Dropped Data

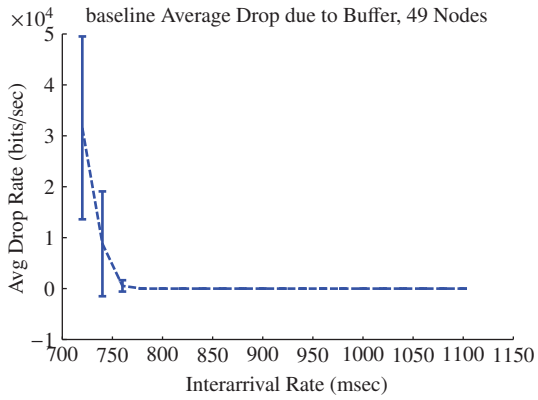


Figure A.31: N49 Buffer Dropped Data

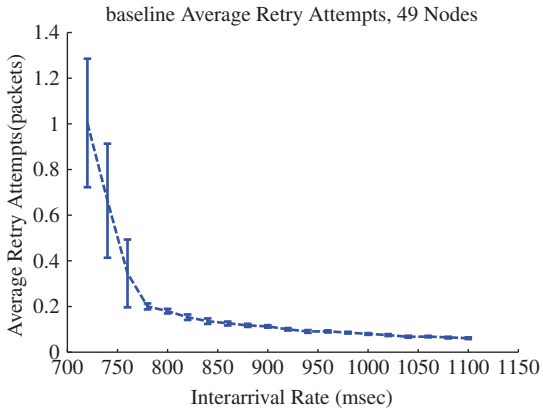


Figure A.32: N49 Retry Attempts

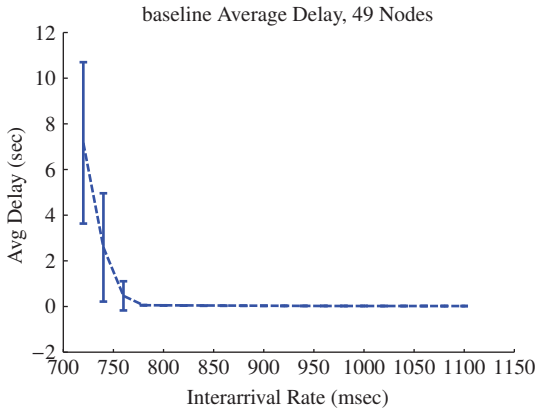


Figure A.33: N49 Packet Delay

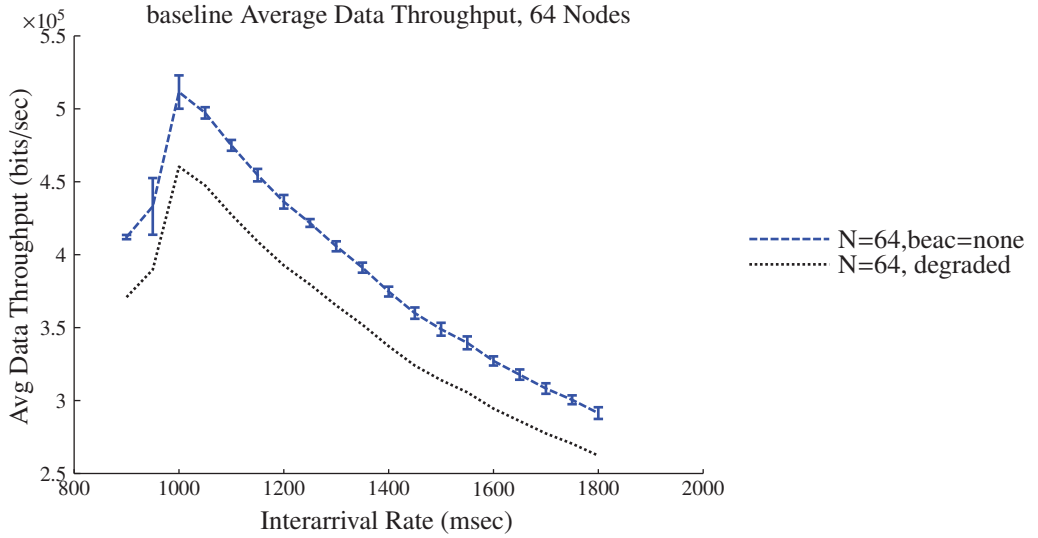


Figure A.34: N64 Data Throughput

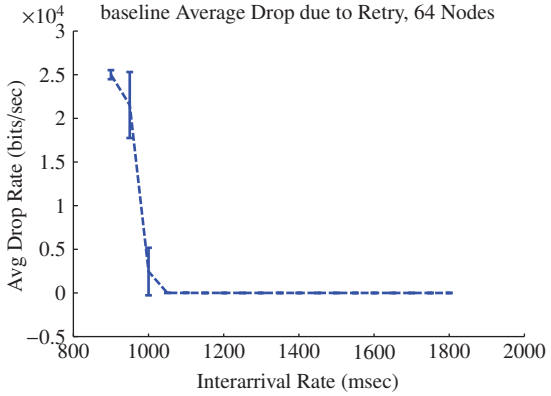


Figure A.35: N64 Retry Dropped Data

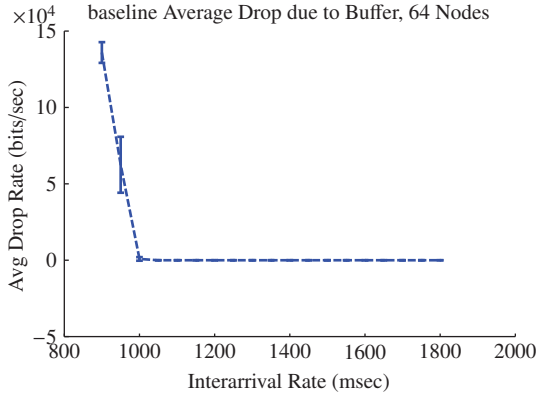


Figure A.36: N64 Buffer Dropped Data

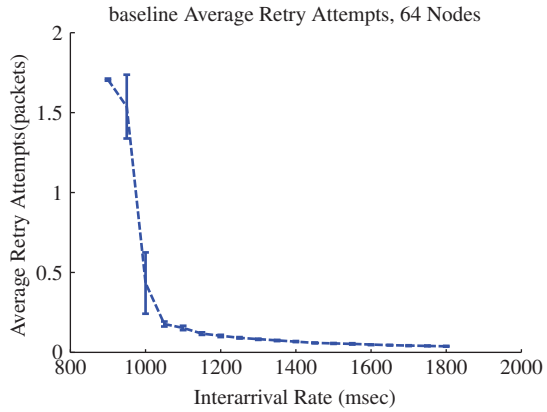


Figure A.37: N64 Retry Attempts

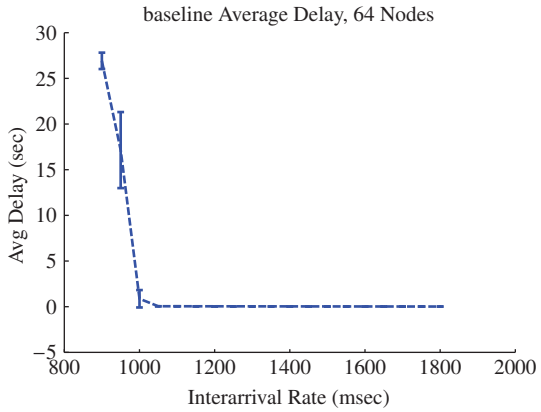


Figure A.38: N64 Packet Delay

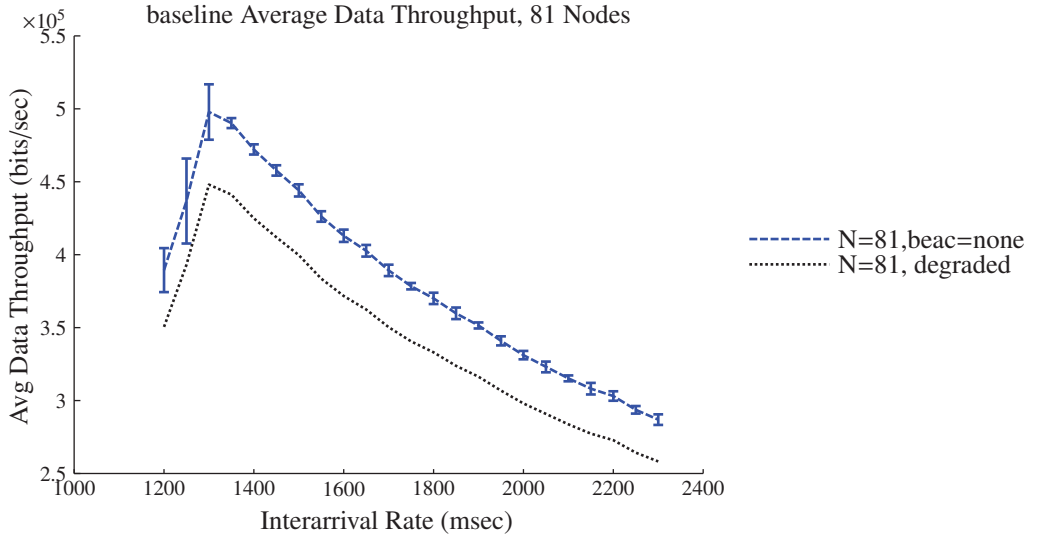


Figure A.39: N81 Data Throughput

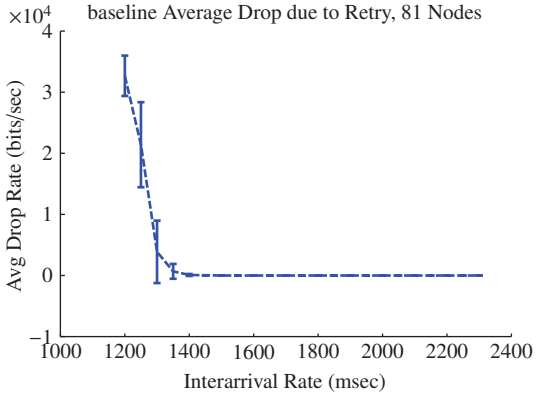


Figure A.40: N81 Retry Dropped Data

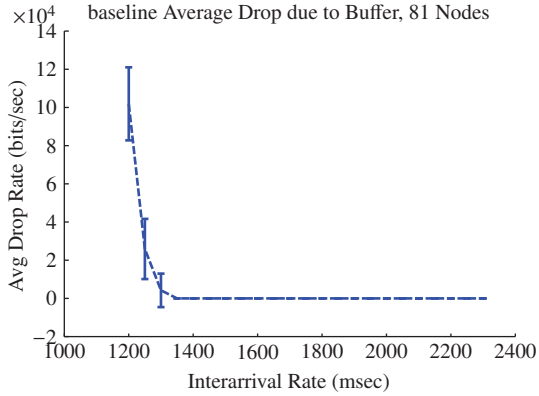


Figure A.41: N81 Buffer Dropped Data

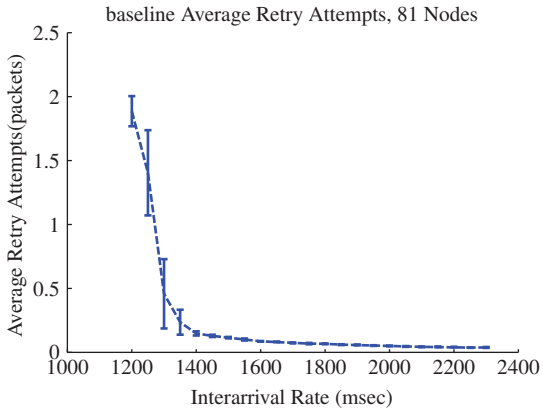


Figure A.42: N81 Retry Attempts

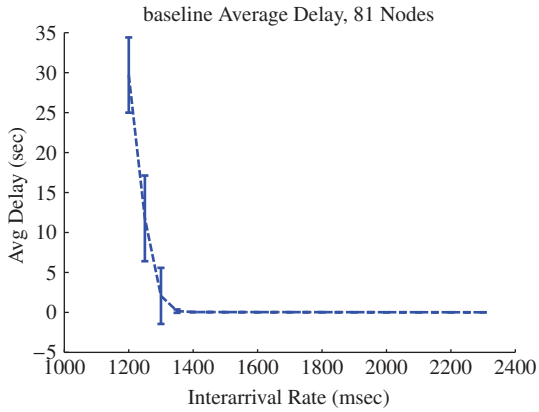


Figure A.43: N81 Packet Delay

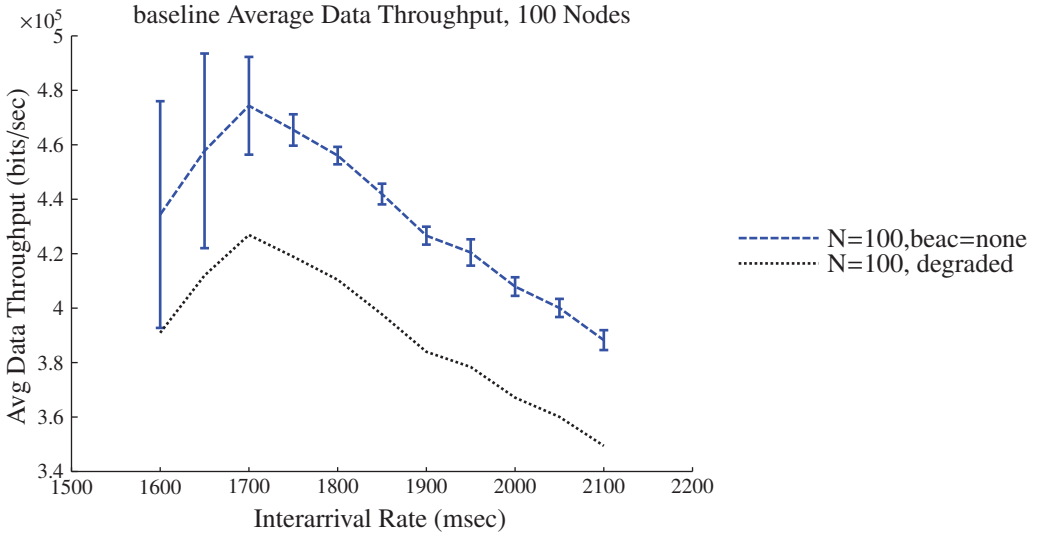


Figure A.44: N100 Data Throughput

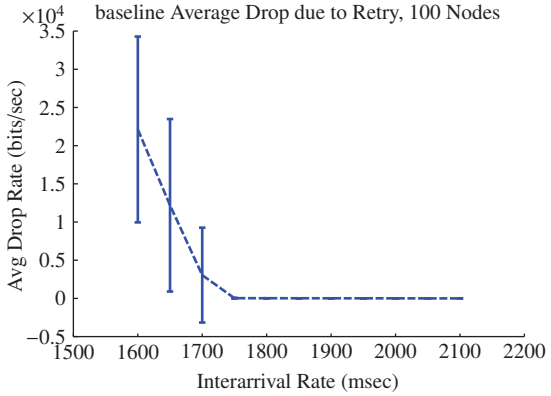


Figure A.45: N100 Retry Dropped Data

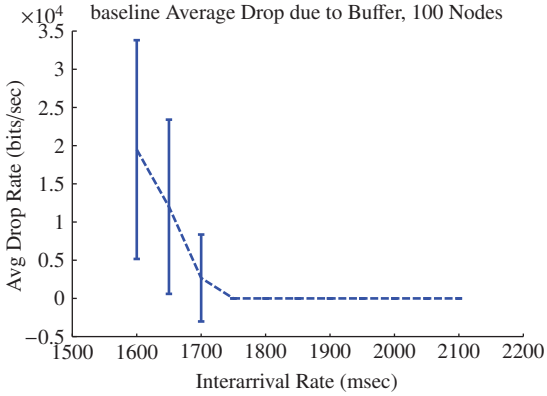


Figure A.46: N100 Buffer Dropped Data

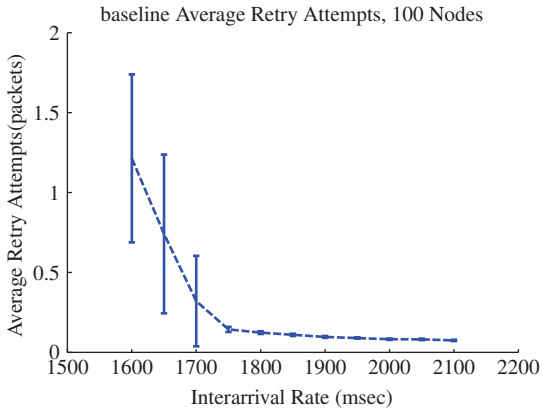


Figure A.47: N100 Retry Attempts

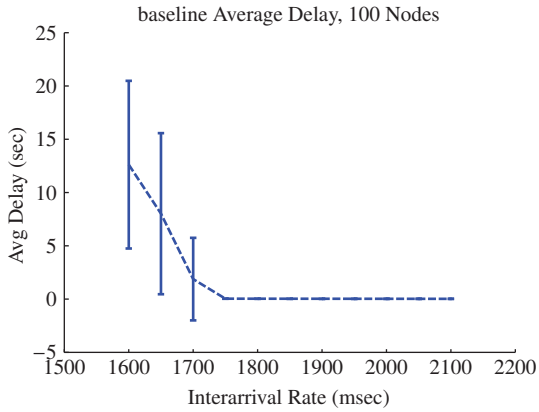


Figure A.48: N100 Packet Delay

A.3 Polling Protocol Results

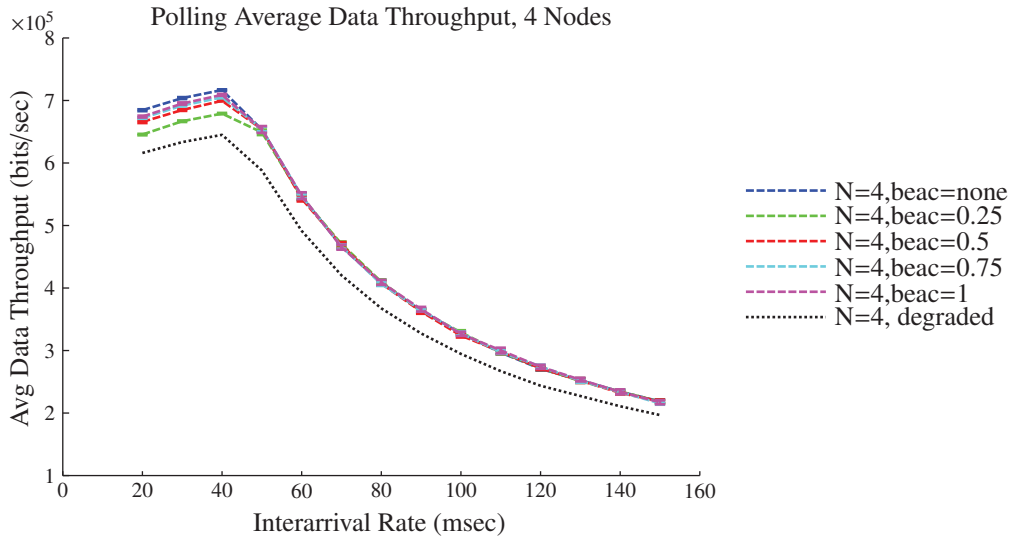


Figure A.49: N4 Data Throughput

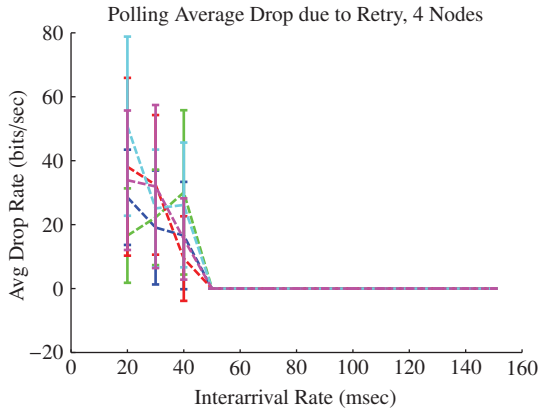


Figure A.50: N4 Retry Dropped Data

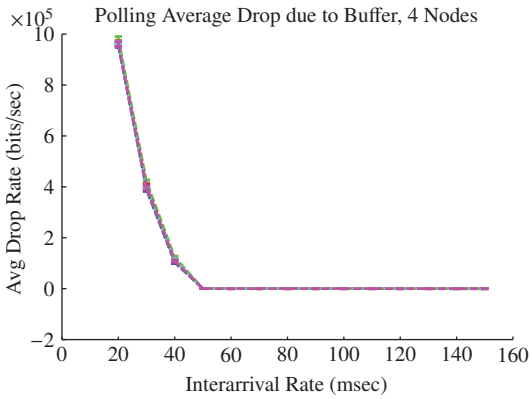


Figure A.51: N4 Buffer Dropped Data

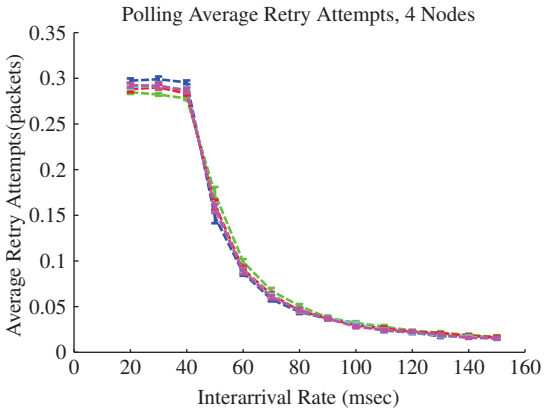


Figure A.52: N4 Retry Attempts

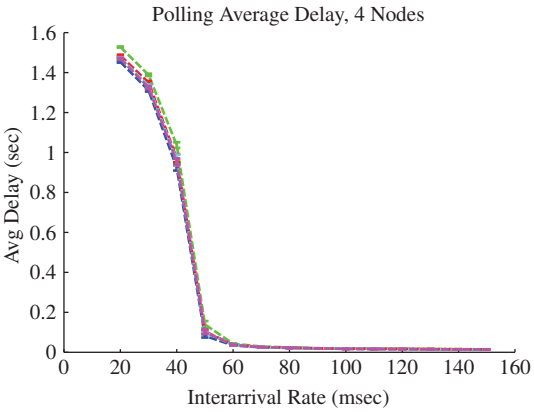


Figure A.53: N4 Packet Delay

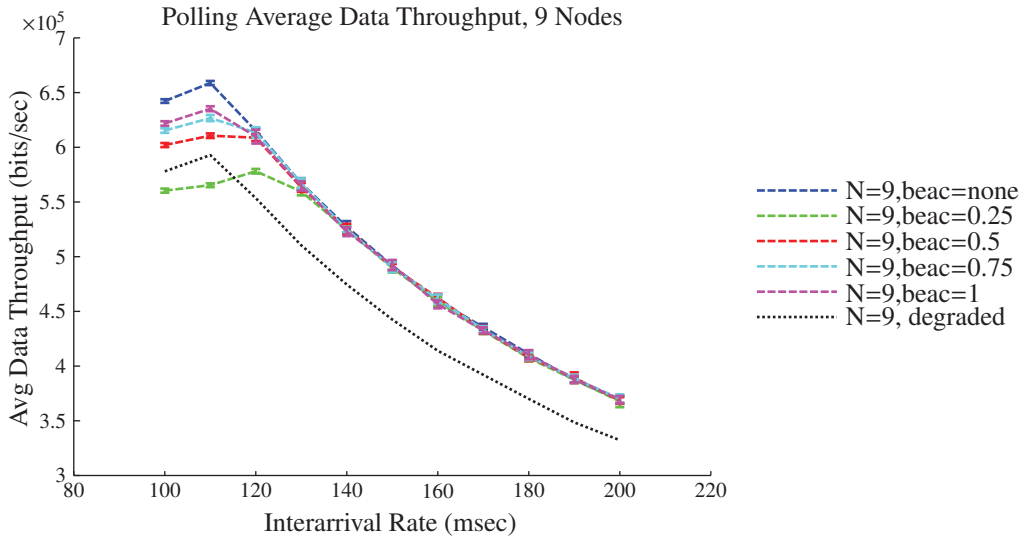


Figure A.54: N9 Data Throughput

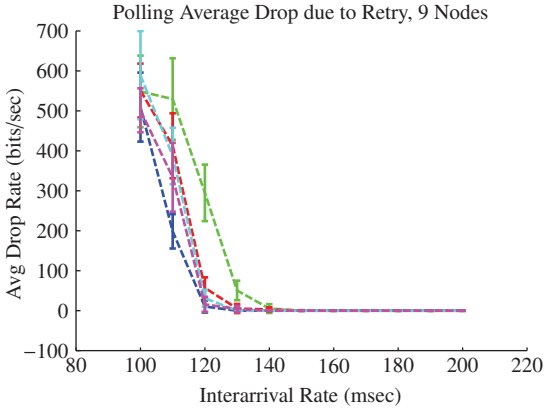


Figure A.55: N9 Retry Dropped Data

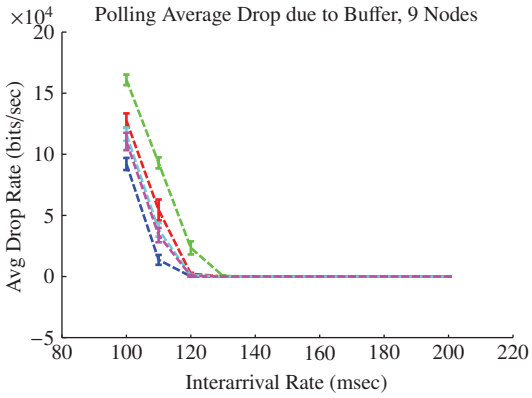


Figure A.56: N9 Buffer Dropped Data

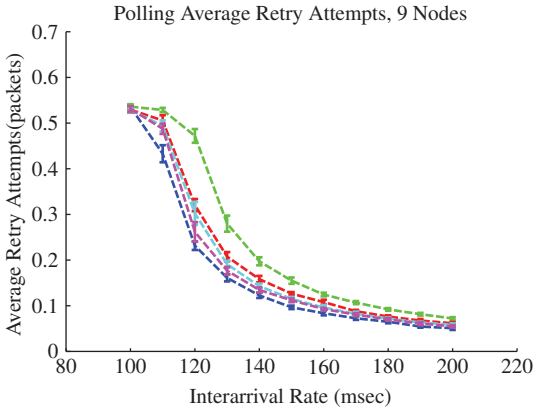


Figure A.57: N9 Retry Attempts

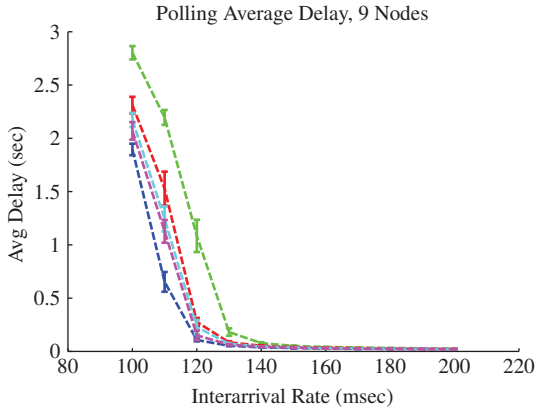


Figure A.58: N9 Packet Delay

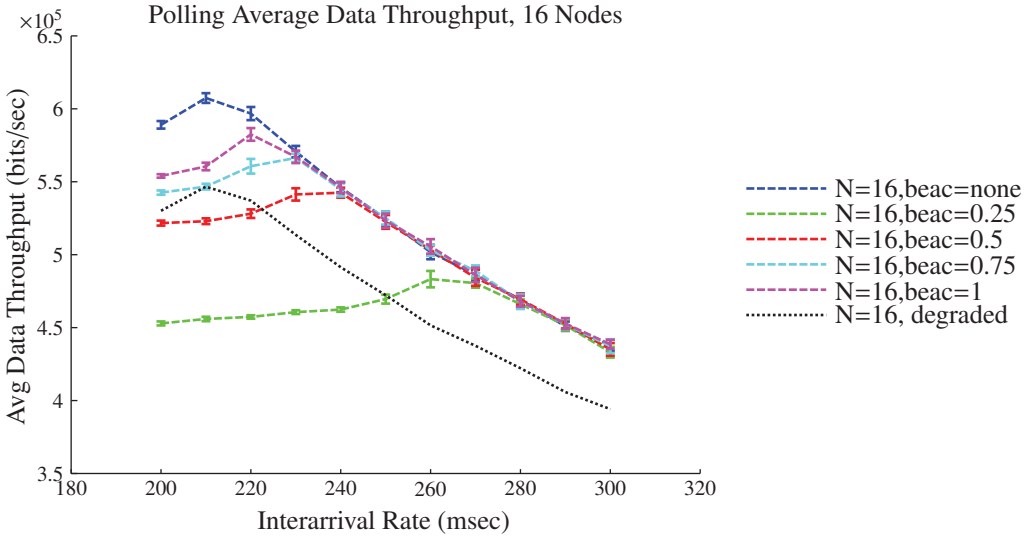


Figure A.59: N16 Data Throughput

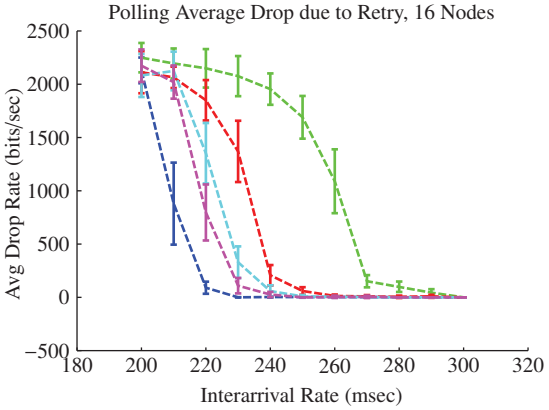


Figure A.60: N16 Retry Dropped Data

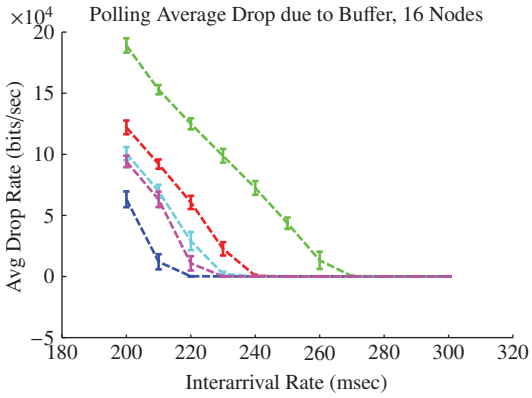


Figure A.61: N16 Buffer Dropped Data

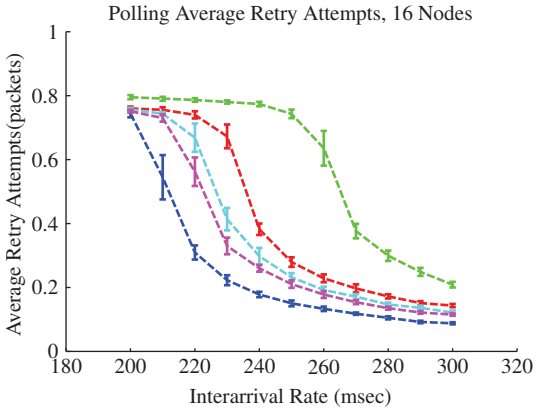


Figure A.62: N16 Retry Attempts

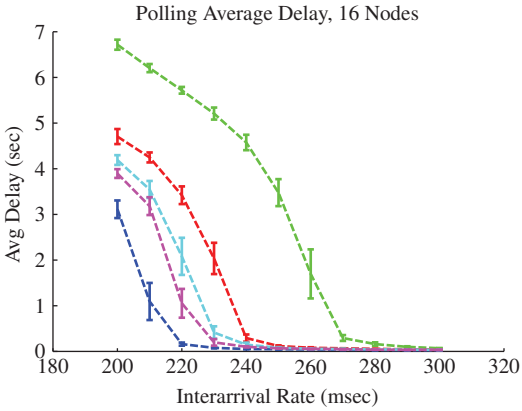


Figure A.63: N16 Packet Delay

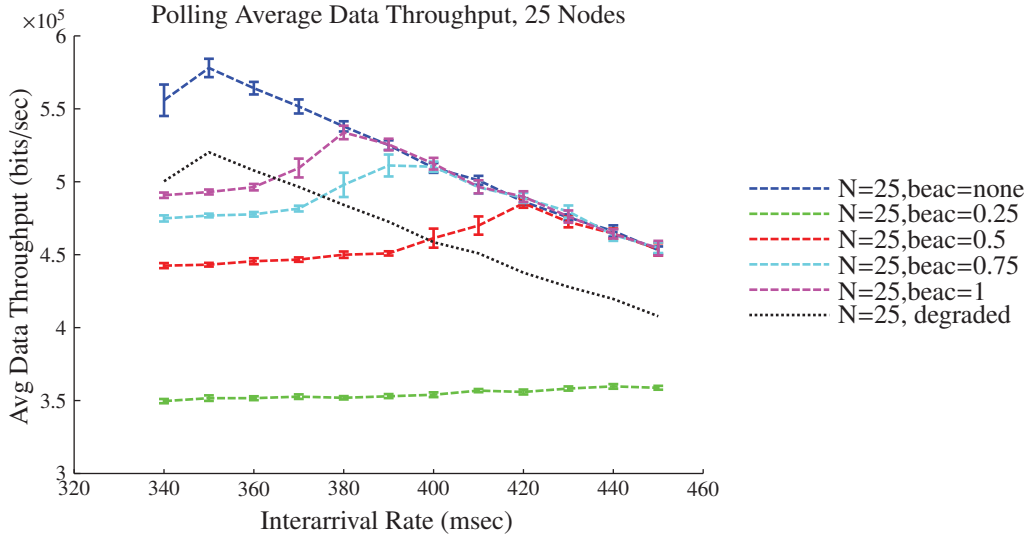


Figure A.64: N25 Data Throughput

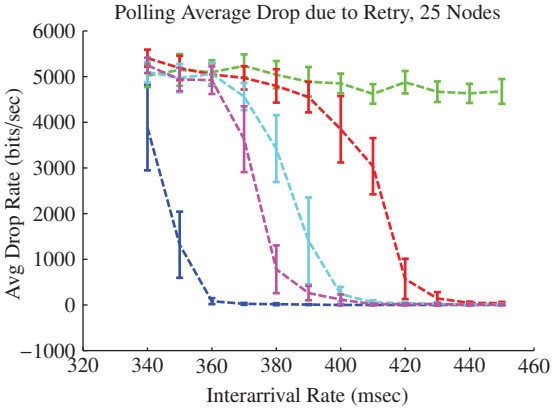


Figure A.65: N25 Retry Dropped Data

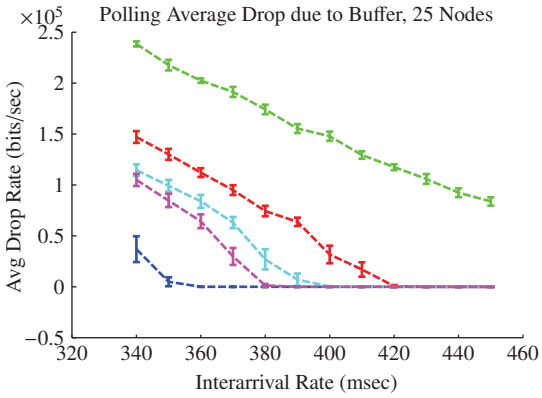


Figure A.66: N25 Buffer Dropped Data

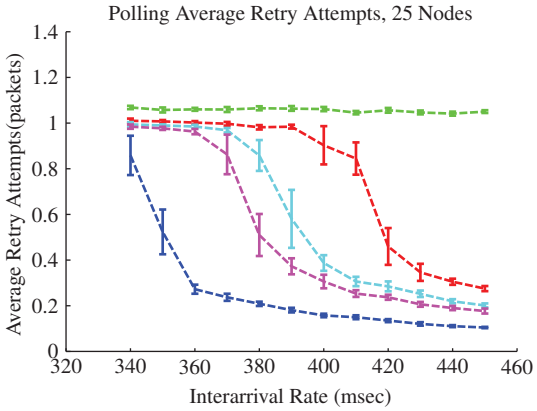


Figure A.67: N25 Retry Attempts

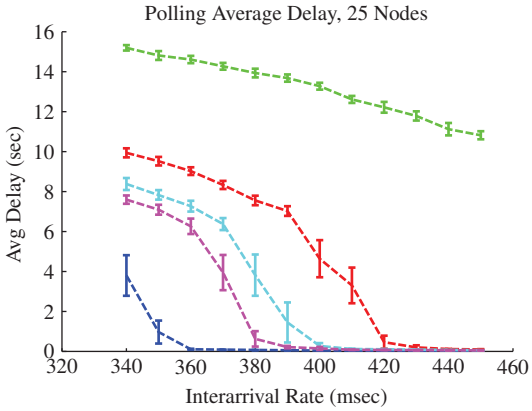


Figure A.68: N25 Packet Delay

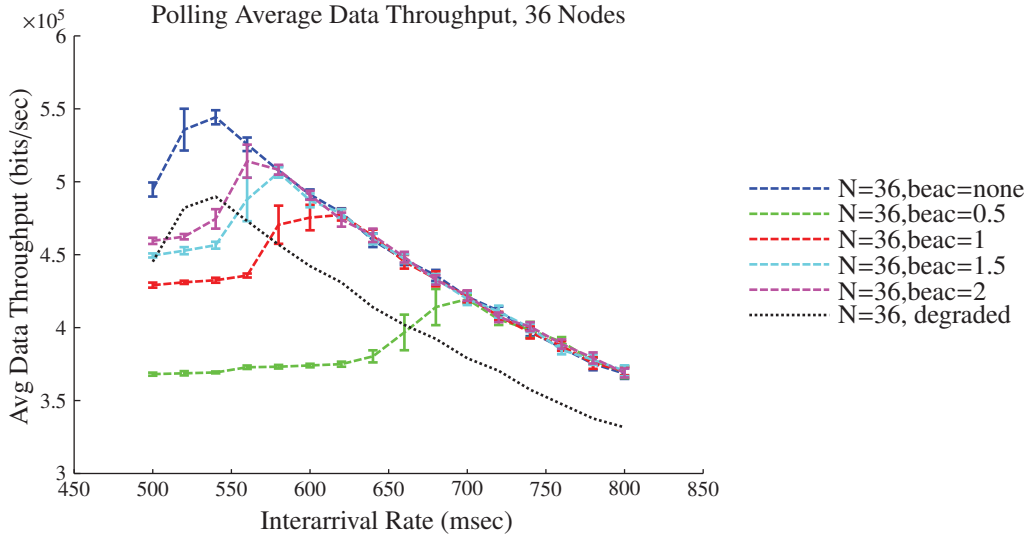


Figure A.69: N36 Data Throughput

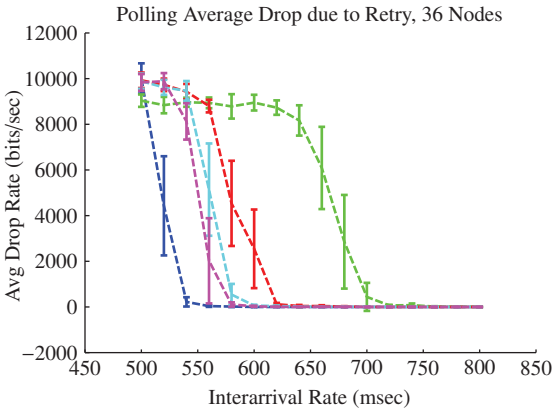


Figure A.70: N36 Retry Dropped Data

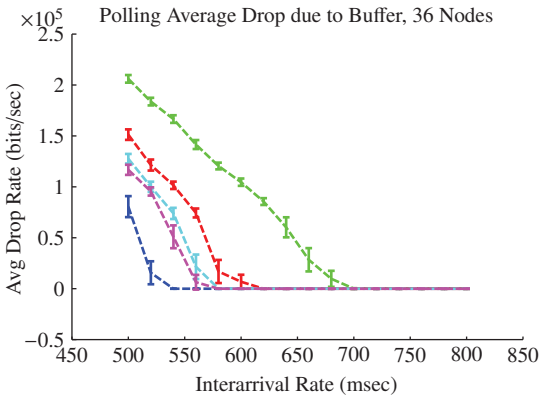


Figure A.71: N36 Buffer Dropped Data

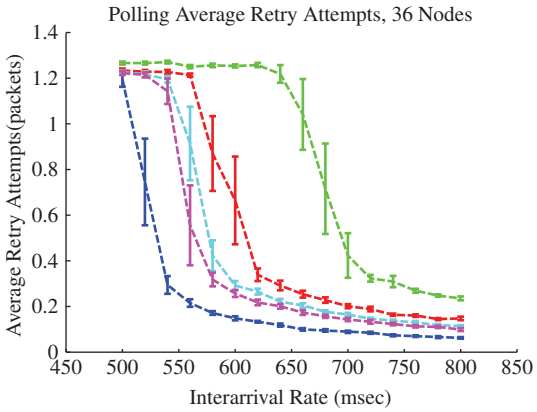


Figure A.72: N36 Retry Attempts

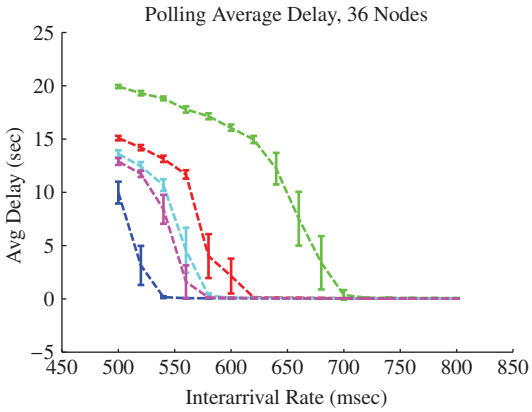


Figure A.73: N36 Packet Delay

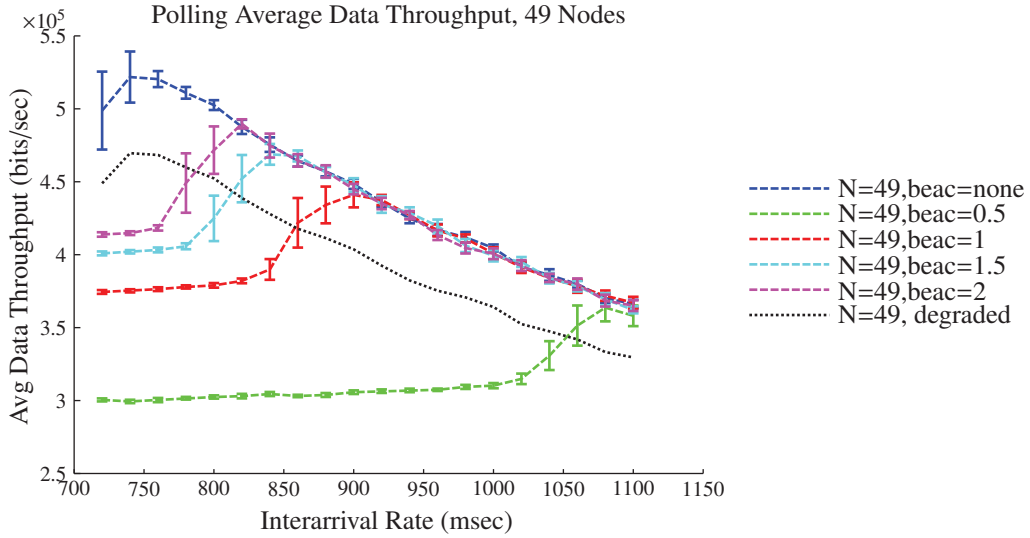


Figure A.74: N49 Data Throughput

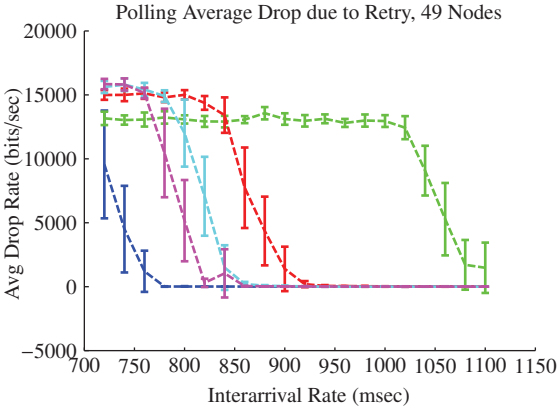


Figure A.75: N49 Retry Dropped Data

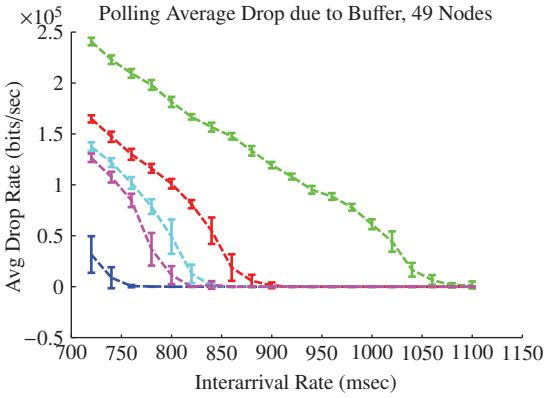


Figure A.76: N49 Buffer Dropped Data

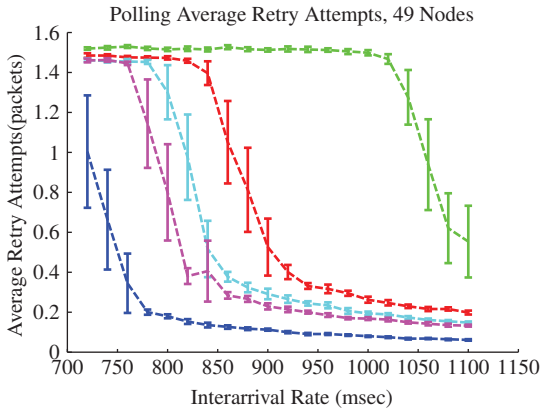


Figure A.77: N49 Retry Attempts

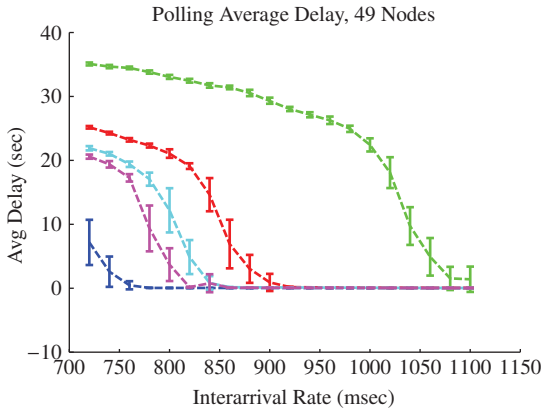


Figure A.78: N49 Packet Delay

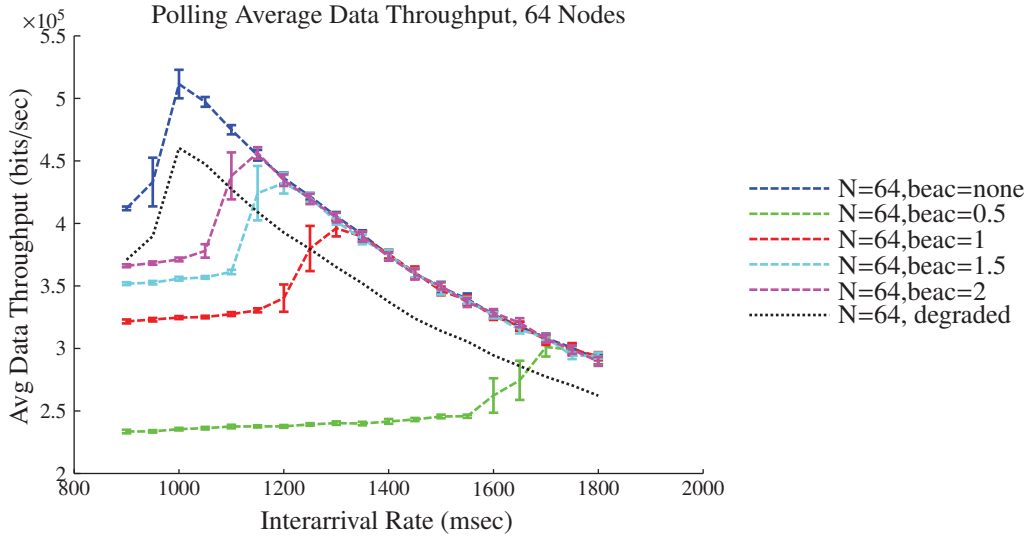


Figure A.79: N64 Data Throughput

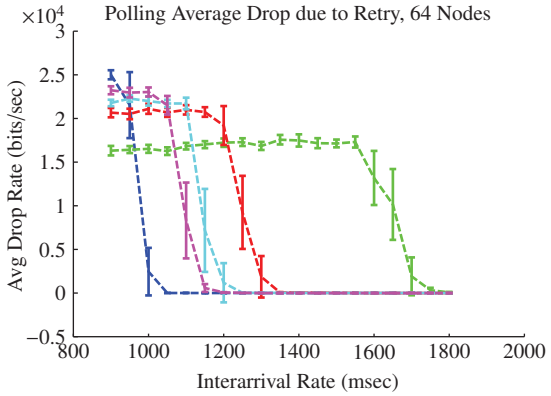


Figure A.80: N64 Retry Dropped Data

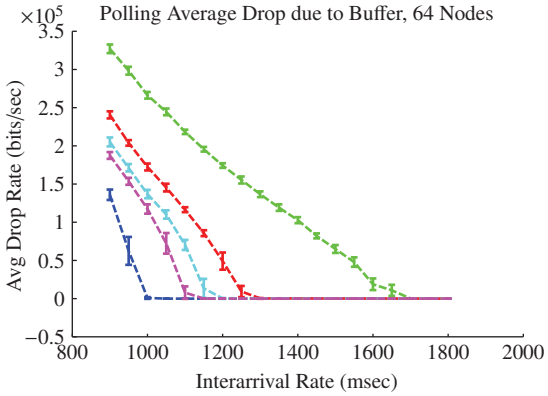


Figure A.81: N64 Buffer Dropped Data

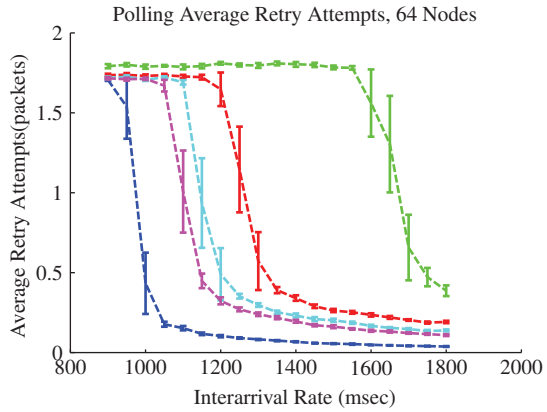


Figure A.82: N64 Retry Attempts

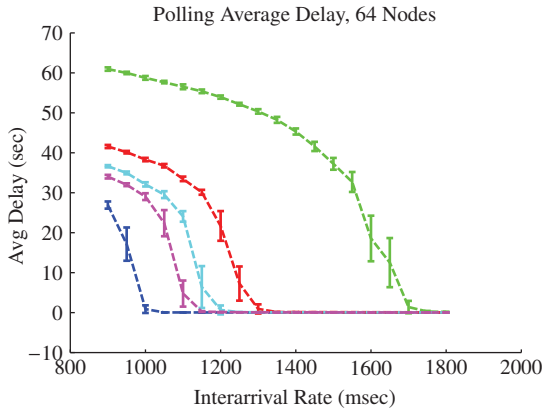


Figure A.83: N64 Packet Delay

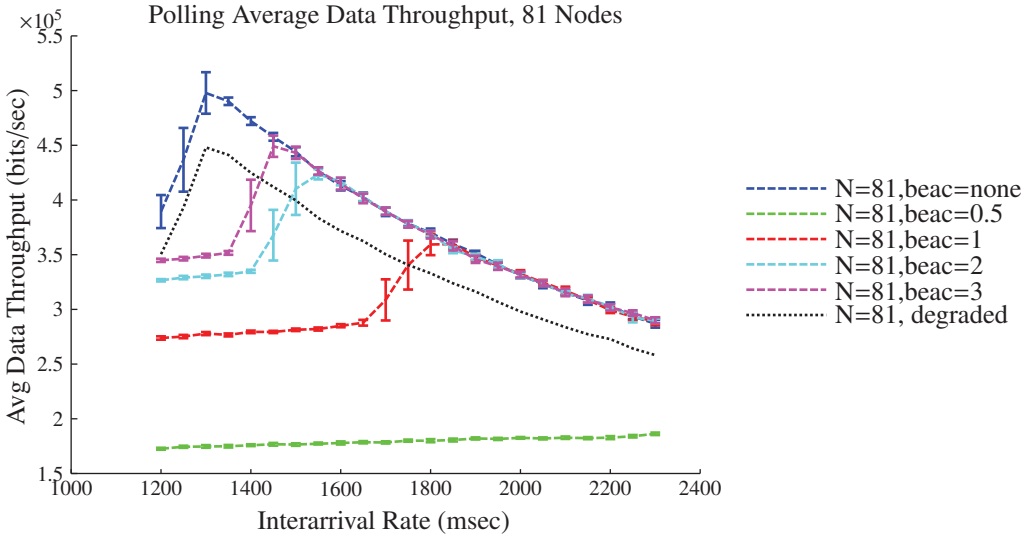


Figure A.84: N81 Data Throughput

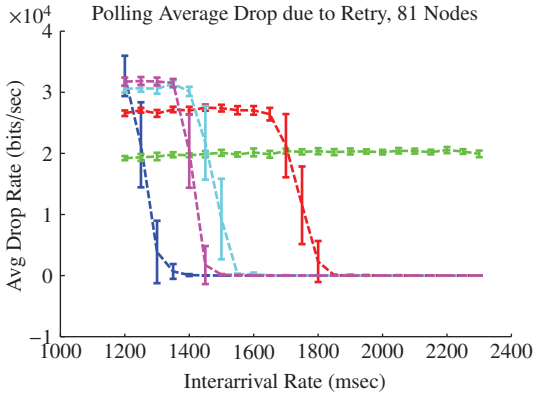


Figure A.85: N81 Retry Dropped Data

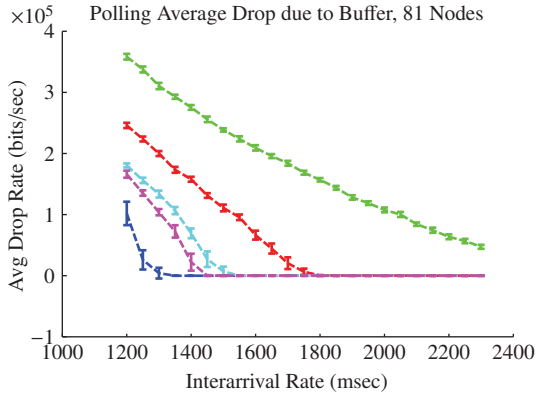


Figure A.86: N81 Buffer Dropped Data

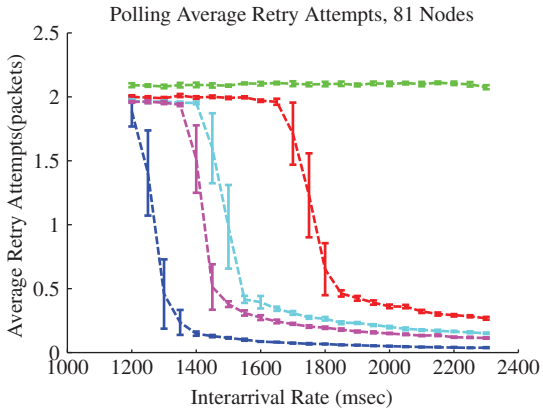


Figure A.87: N81 Retry Attempts

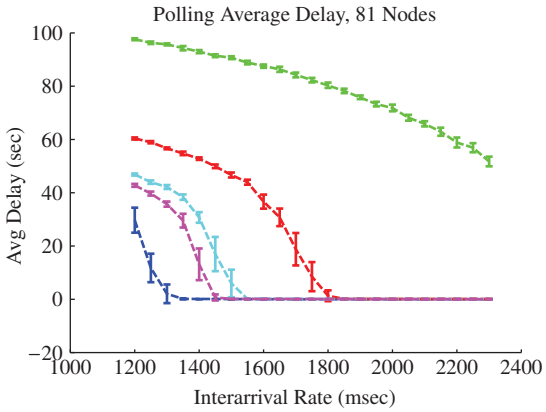


Figure A.88: N81 Packet Delay

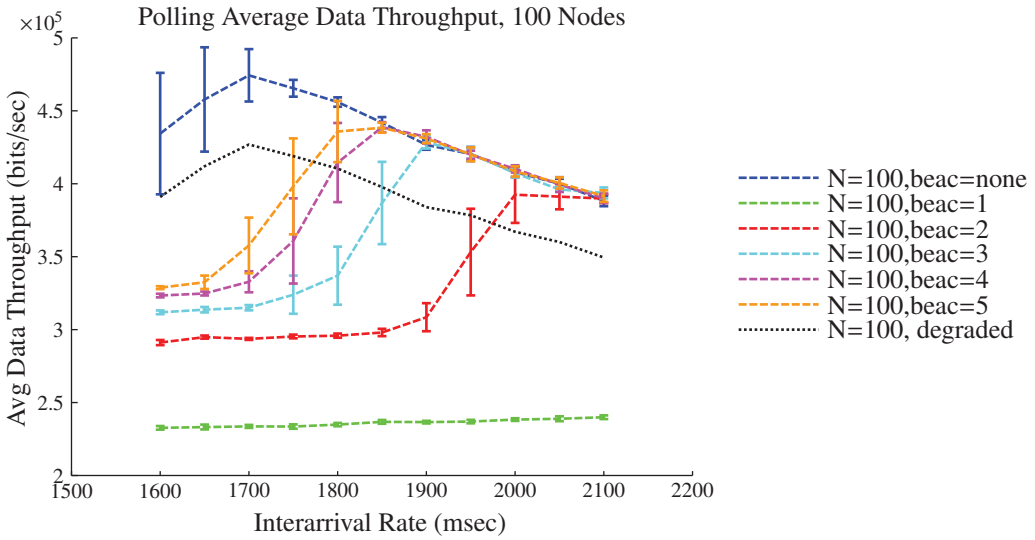


Figure A.89: N100 Data Throughput

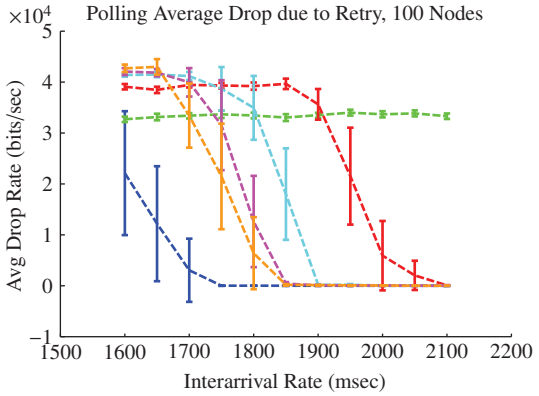


Figure A.90: N100 Retry Dropped Data

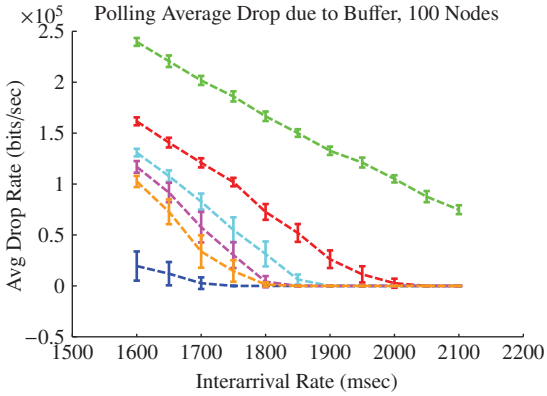


Figure A.91: N100 Buffer Dropped Data

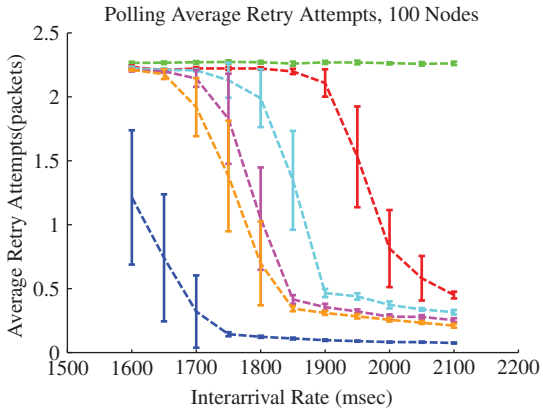


Figure A.92: N100 Retry Attempts

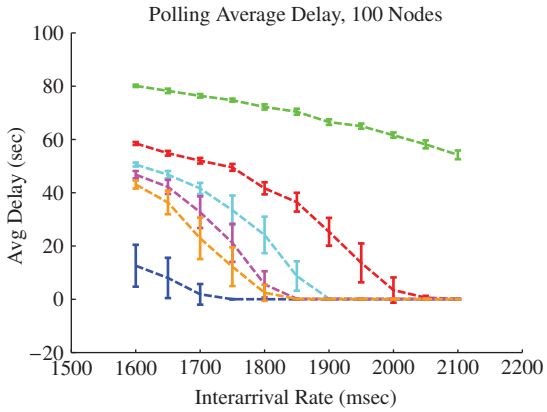


Figure A.93: N100 Packet Delay

A.4 Time Division Protocol Results

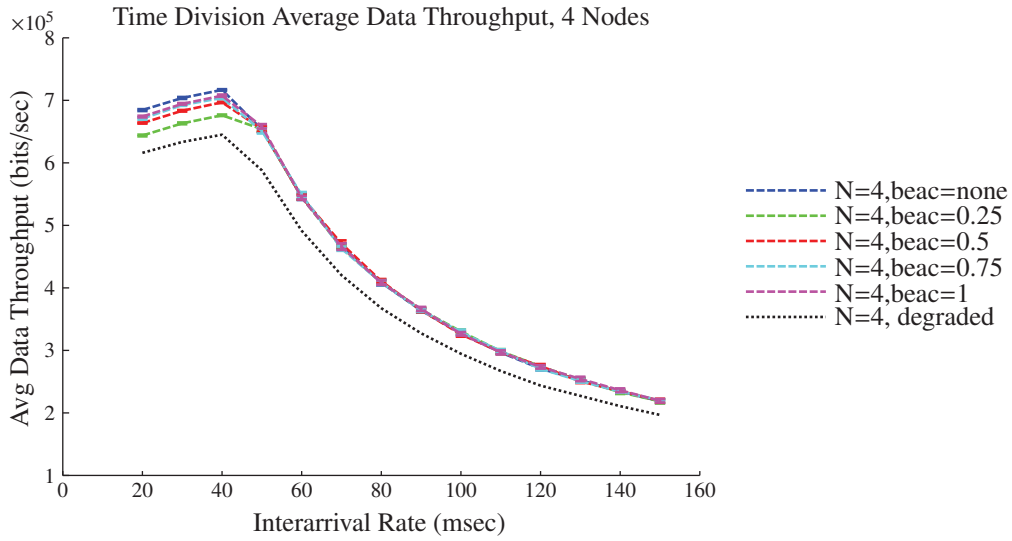


Figure A.94: N4 Data Throughput

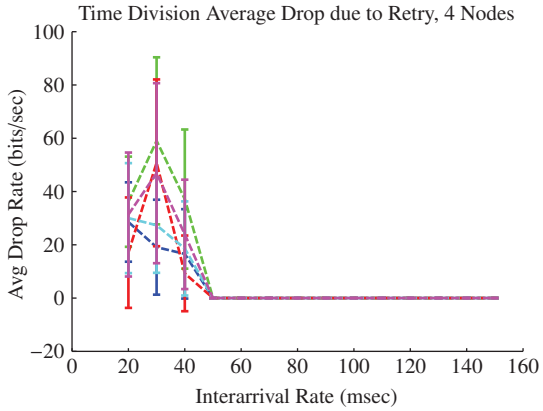


Figure A.95: N4 Retry Dropped Data

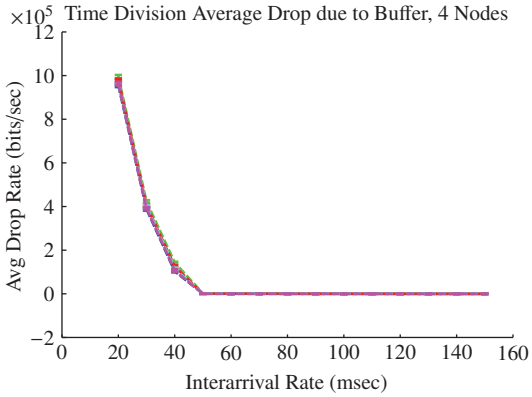


Figure A.96: N4 Buffer Dropped Data

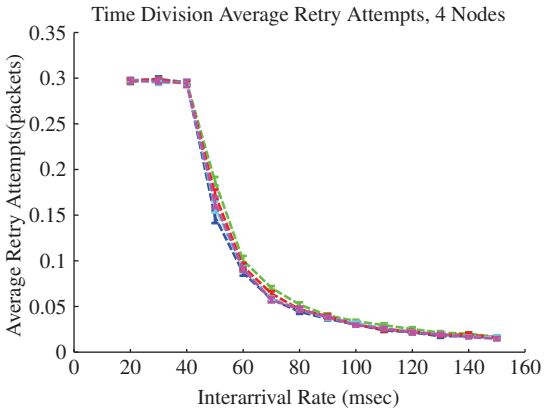


Figure A.97: N4 Retry Attempts

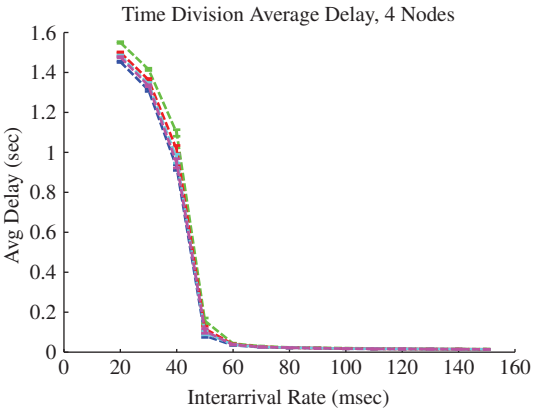


Figure A.98: N4 Packet Delay

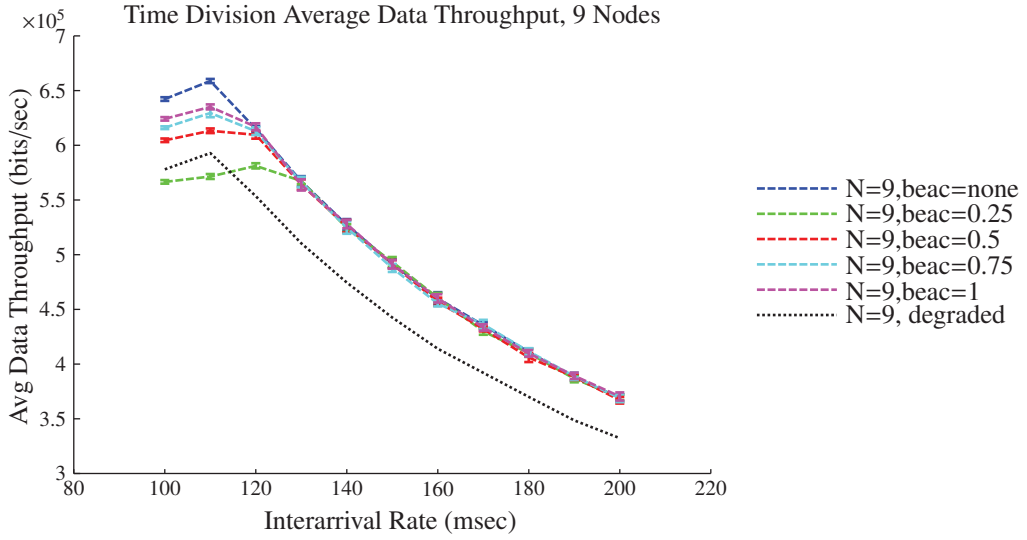


Figure A.99: N9 Data Throughput

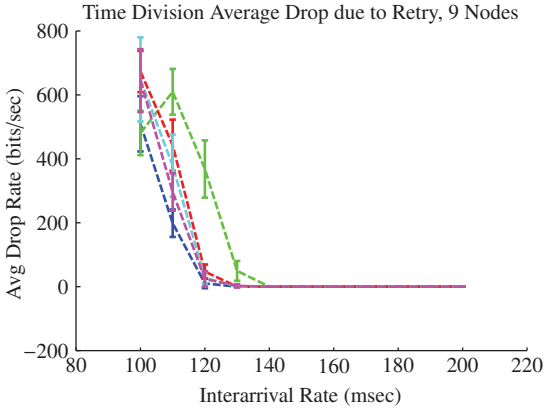


Figure A.100: N9 Retry Dropped Data

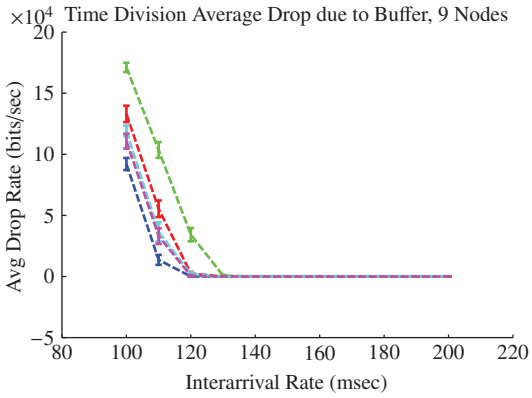


Figure A.101: N9 Buffer Dropped Data

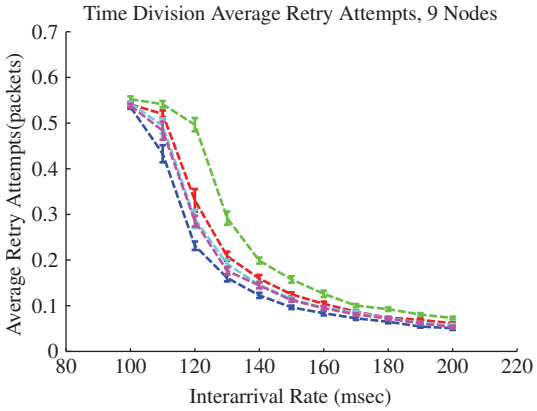


Figure A.102: N9 Retry Attempts

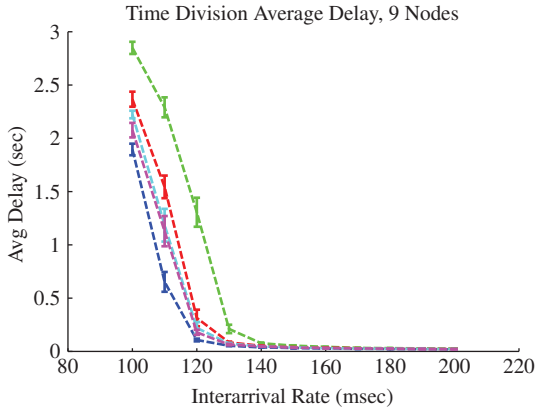


Figure A.103: N9 Packet Delay

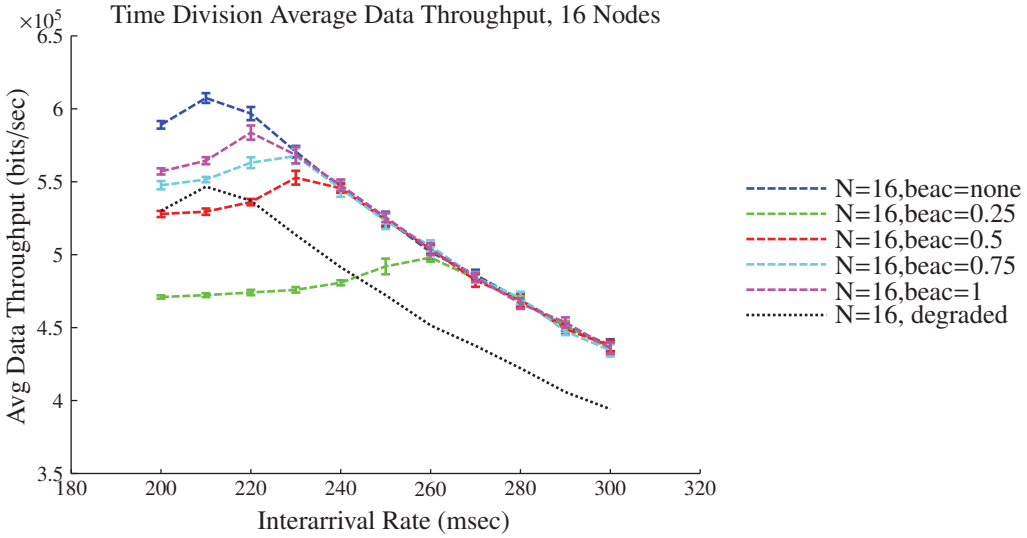


Figure A.104: N16 Data Throughput

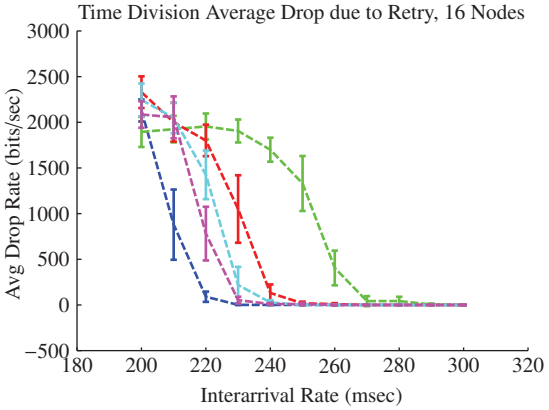


Figure A.105: N16 Retry Dropped Data

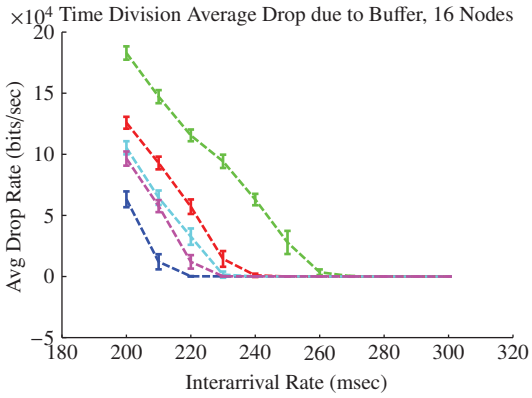


Figure A.106: N16 Buffer Dropped Data

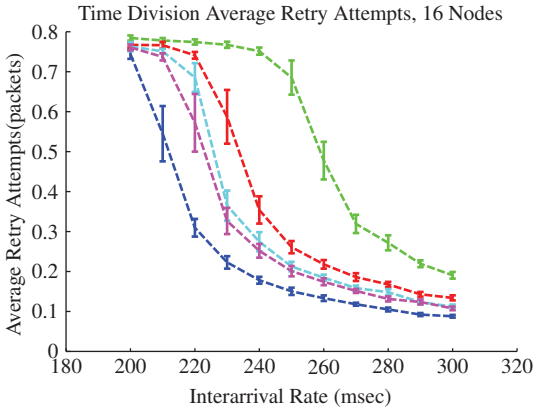


Figure A.107: N16 Retry Attempts

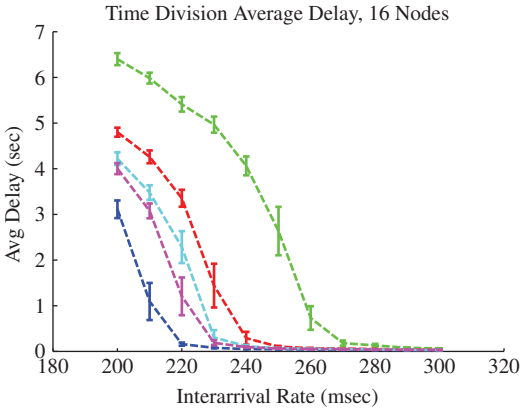


Figure A.108: N16 Packet Delay

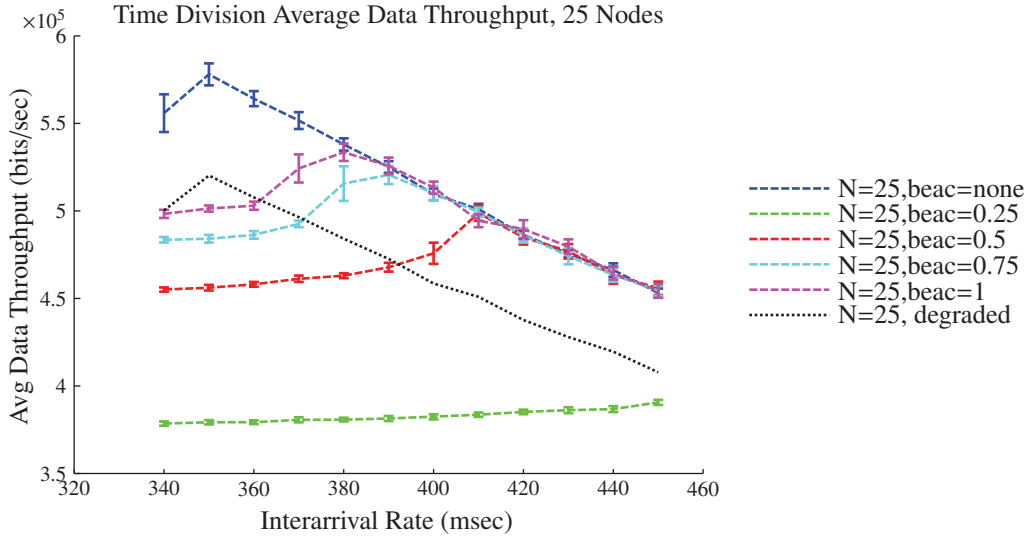


Figure A.109: N25 Data Throughput

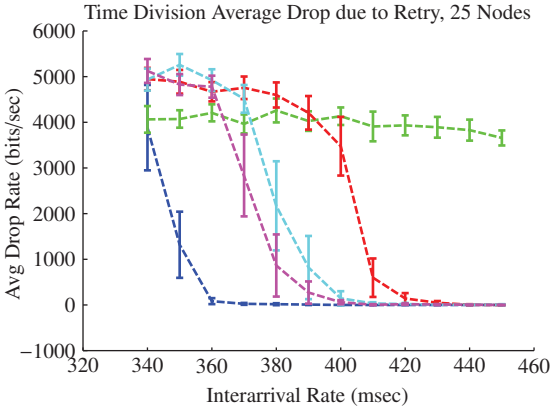


Figure A.110: N25 Retry Dropped Data

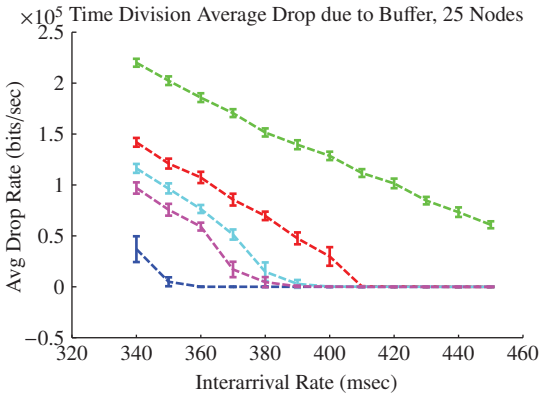


Figure A.111: N25 Buffer Dropped Data

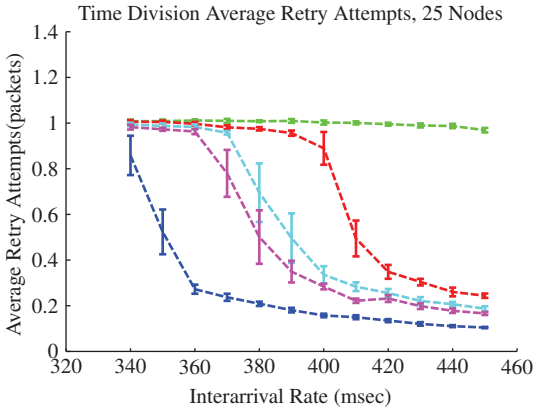


Figure A.112: N25 Retry Attempts

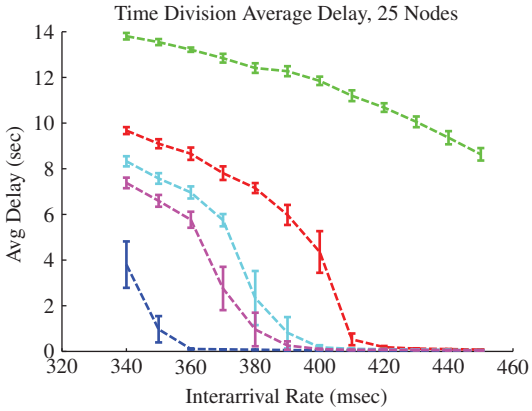


Figure A.113: N25 Packet Delay

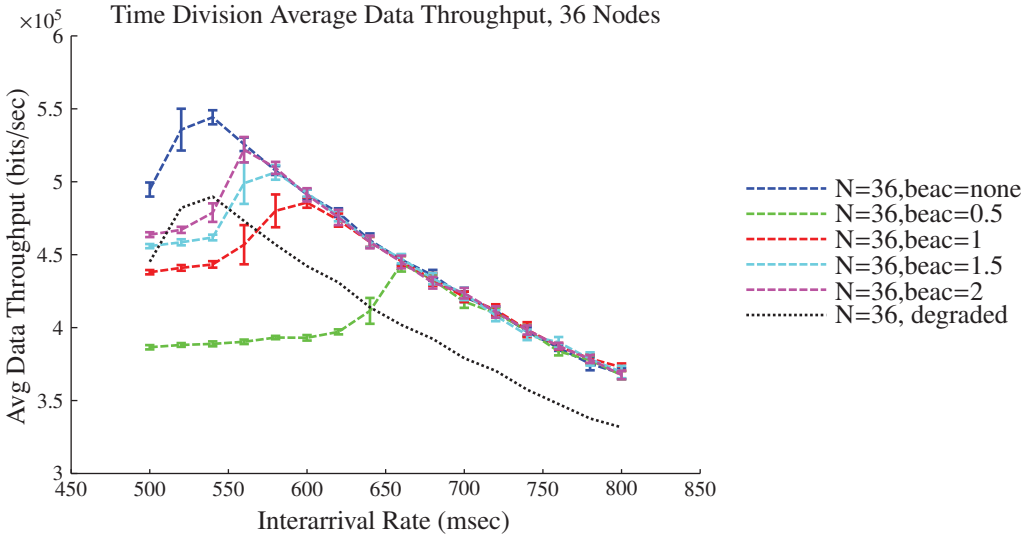


Figure A.114: N36 Data Throughput

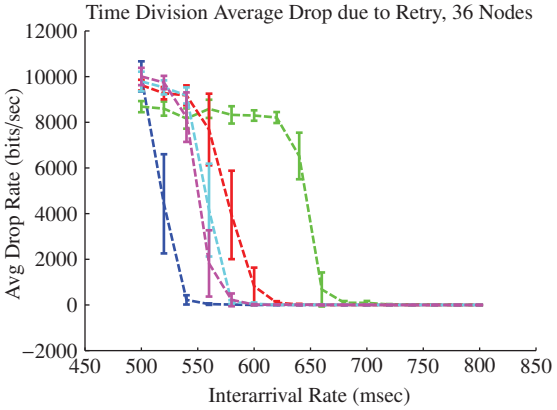


Figure A.115: N36 Retry Dropped Data

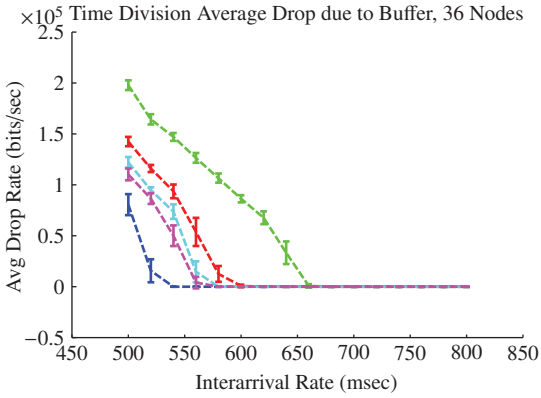


Figure A.116: N36 Buffer Dropped Data

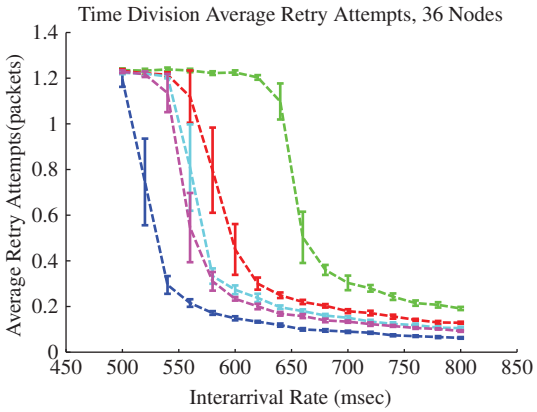


Figure A.117: N36 Retry Attempts

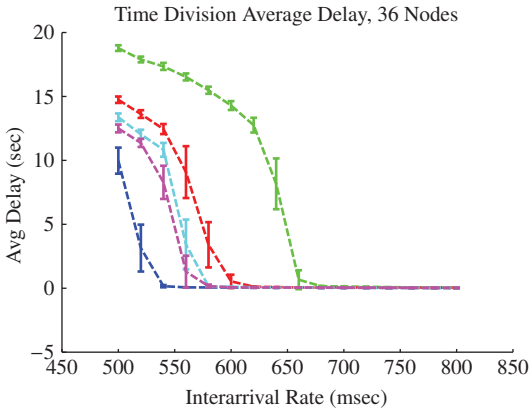


Figure A.118: N36 Packet Delay

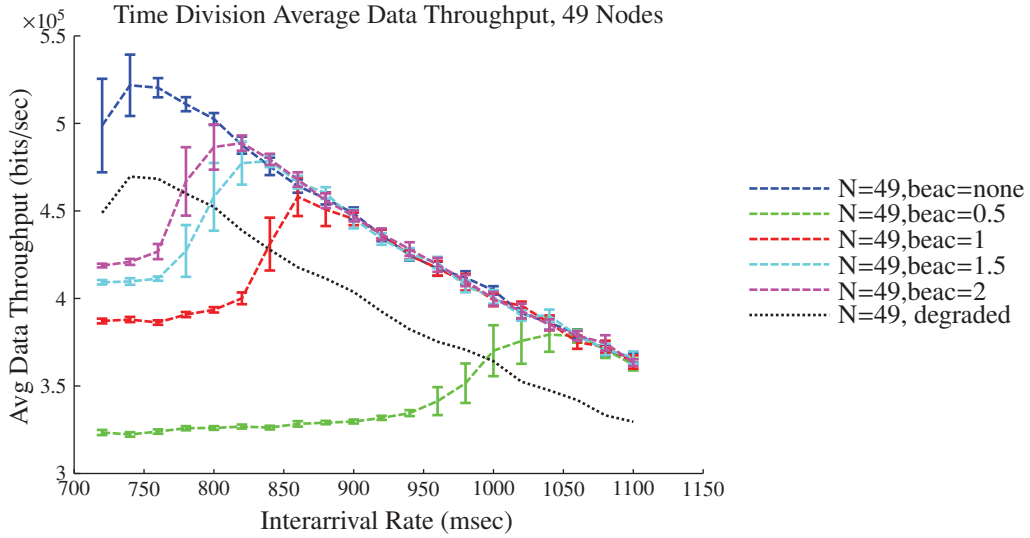


Figure A.119: N49 Data Throughput

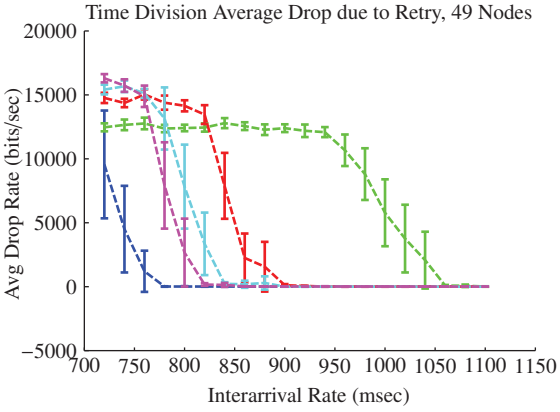


Figure A.120: N49 Retry Dropped Data

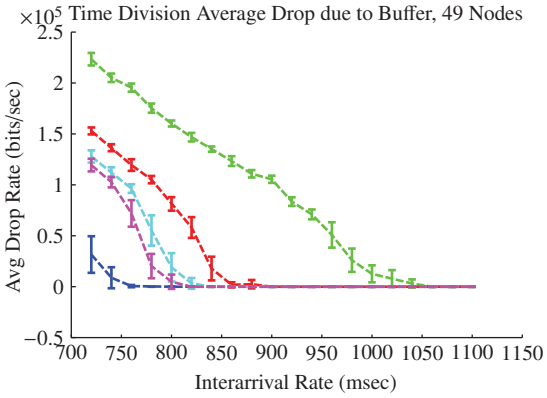


Figure A.121: N49 Buffer Dropped Data

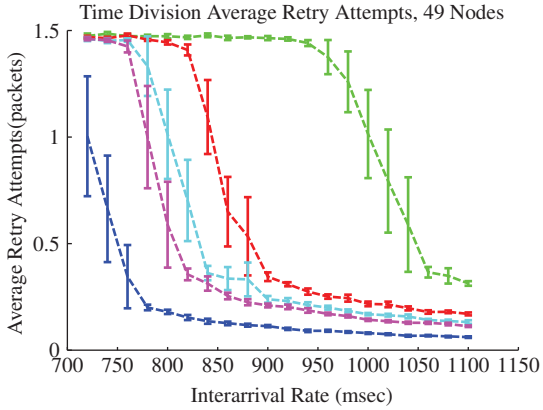


Figure A.122: N49 Retry Attempts

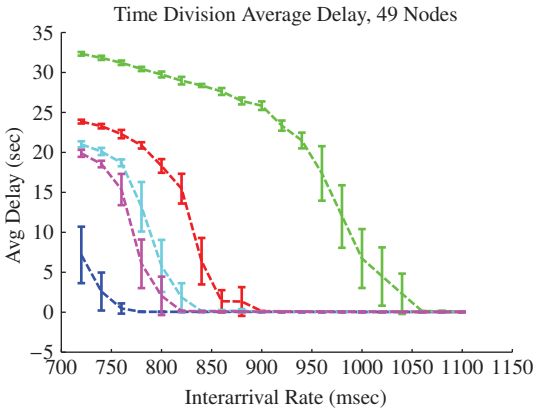


Figure A.123: N49 Packet Delay

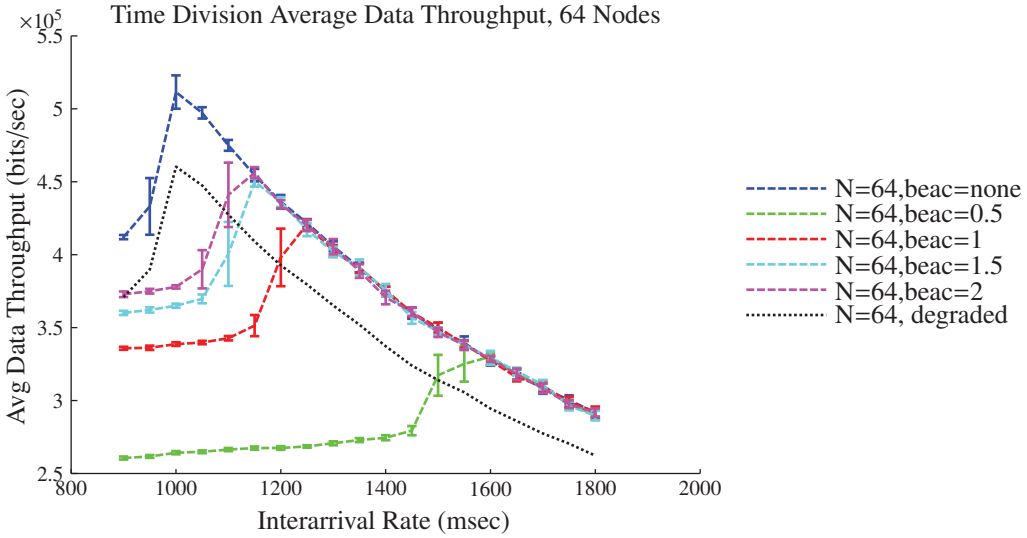


Figure A.124: N64 Data Throughput

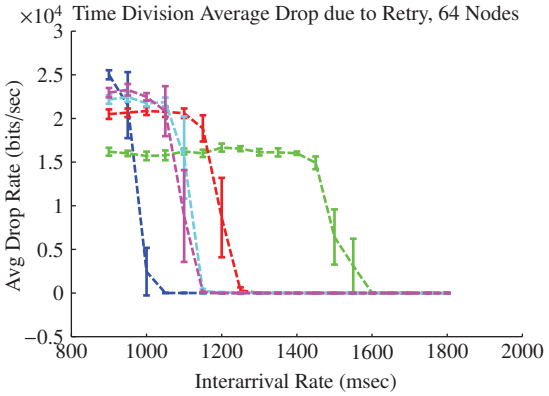


Figure A.125: N64 Retry Dropped Data

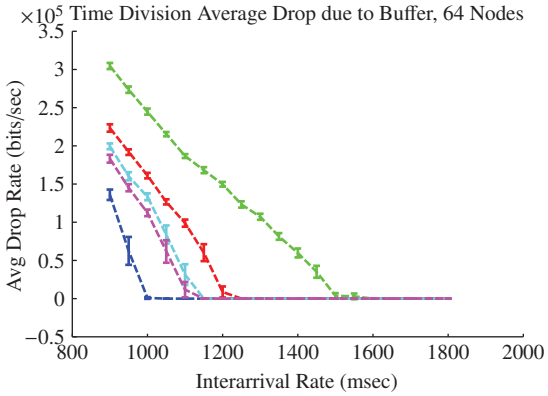


Figure A.126: N64 Buffer Dropped Data

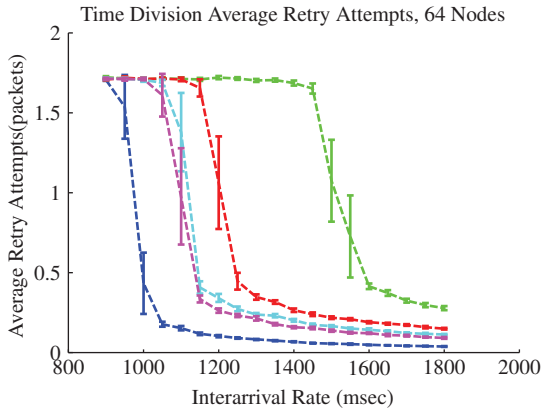


Figure A.127: N64 Retry Attempts

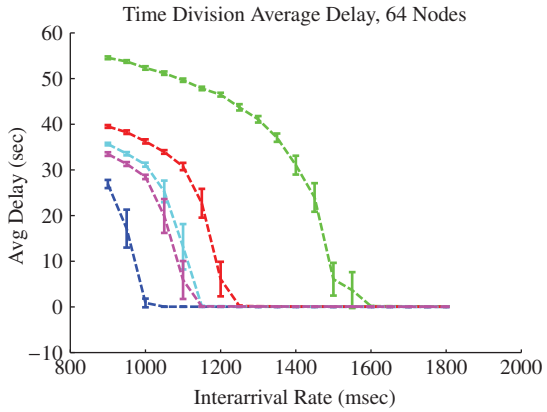


Figure A.128: N64 Packet Delay

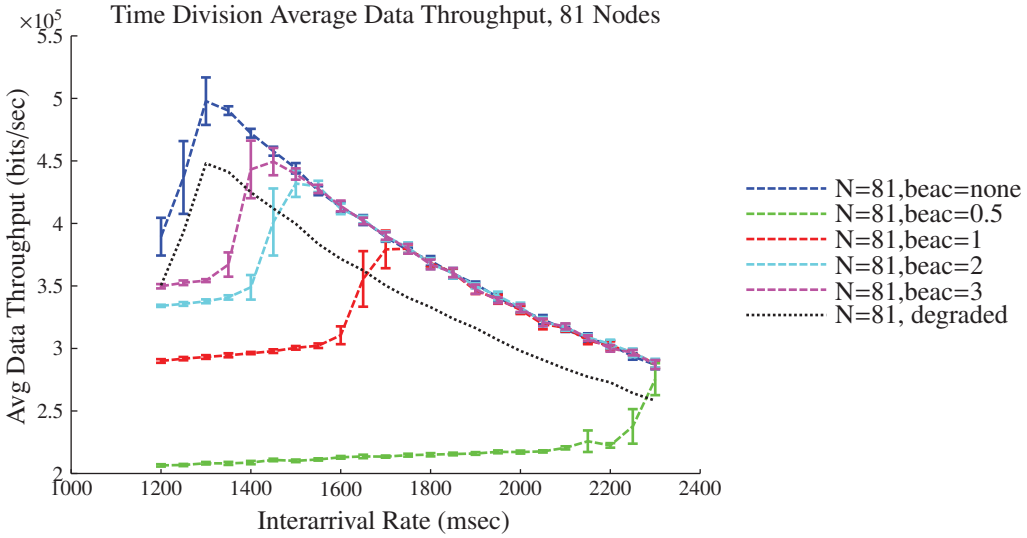


Figure A.129: N81 Data Throughput

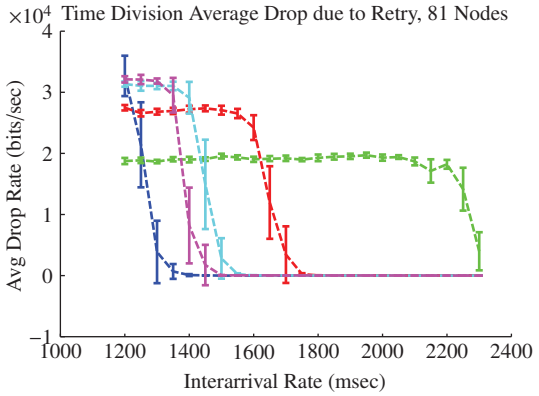


Figure A.130: N81 Retry Dropped Data

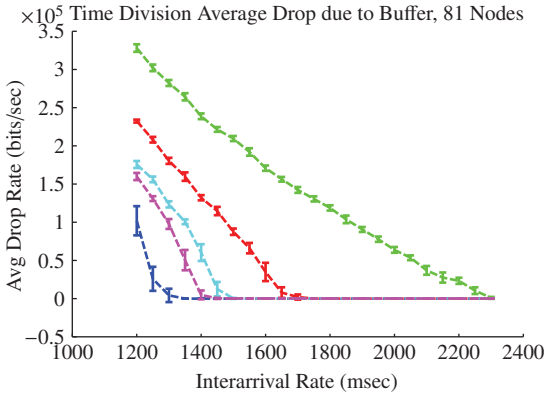


Figure A.131: N81 Buffer Dropped Data

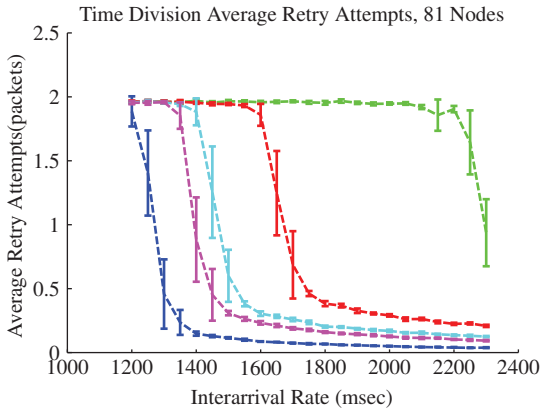


Figure A.132: N81 Retry Attempts

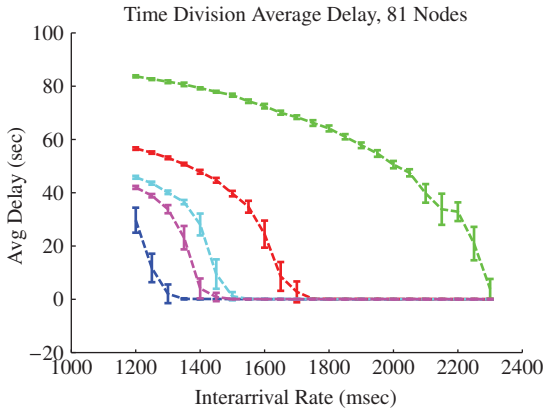


Figure A.133: N81 Packet Delay

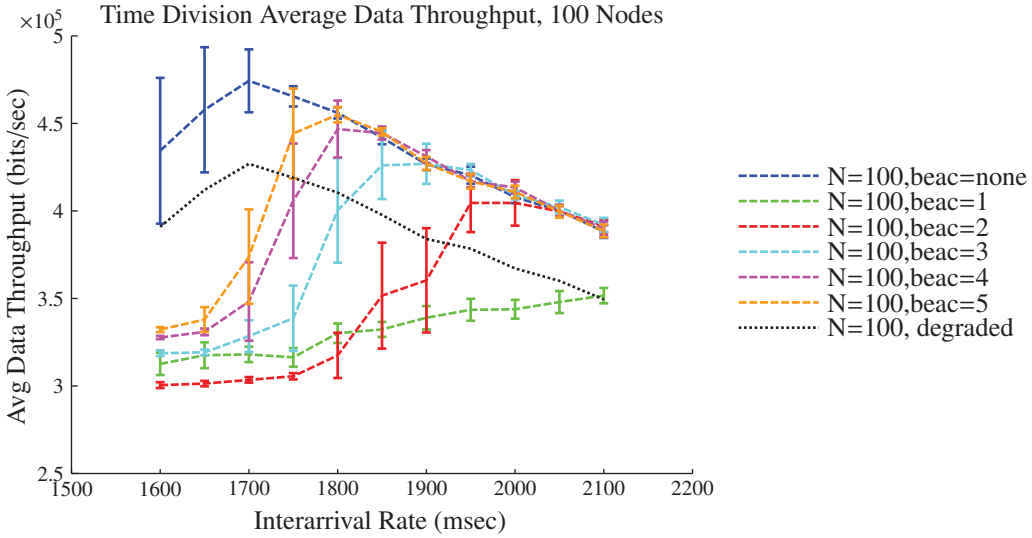


Figure A.134: N100 Data Throughput

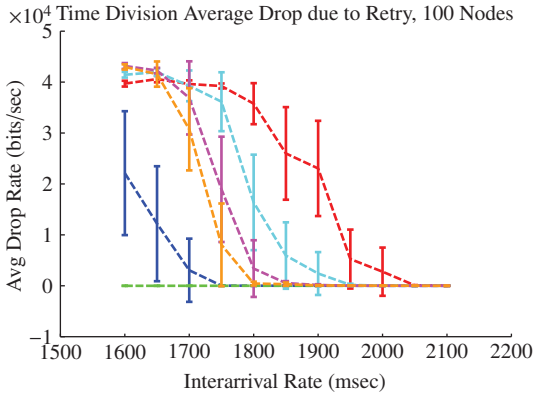


Figure A.135: N100 Retry Dropped Data

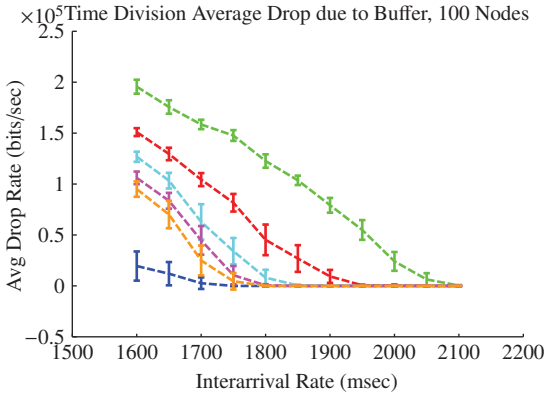


Figure A.136: N100 Buffer Dropped Data

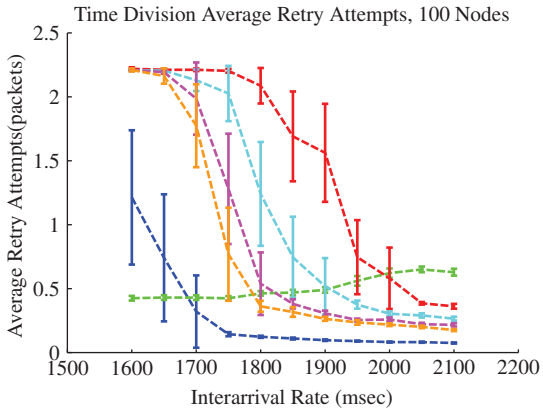


Figure A.137: N100 Retry Attempts

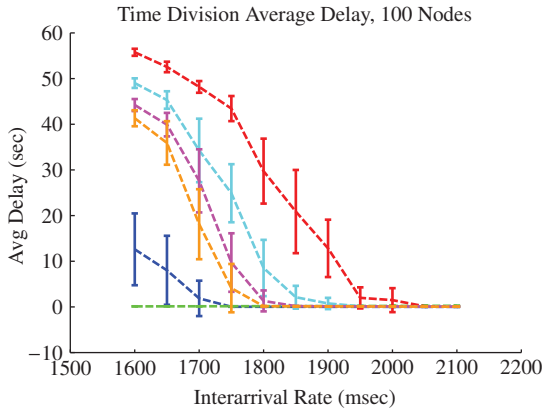


Figure A.138: N100 Packet Delay

A.5 Priority Protocol Results

A.5.1 Unique Scenario.

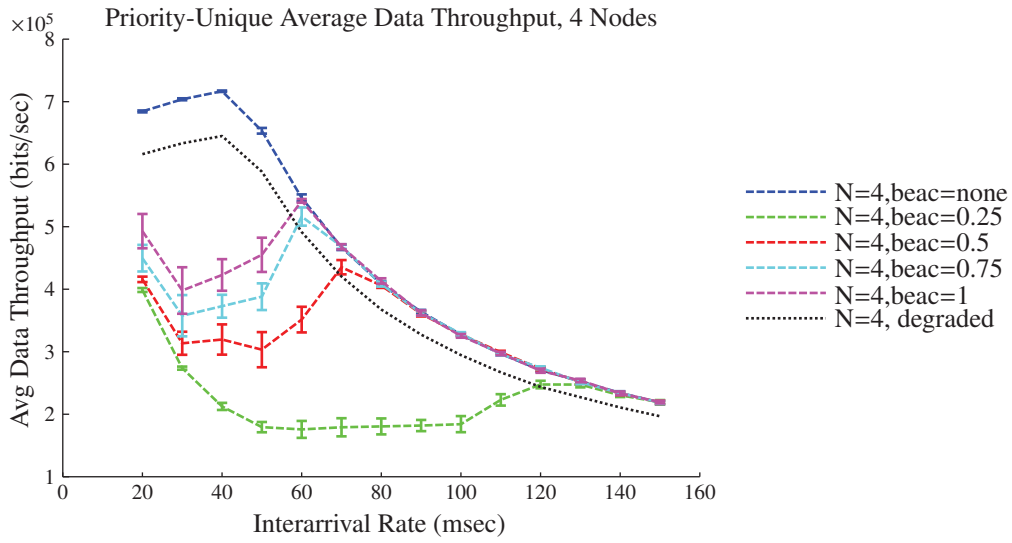


Figure A.139: N4 Data Throughput

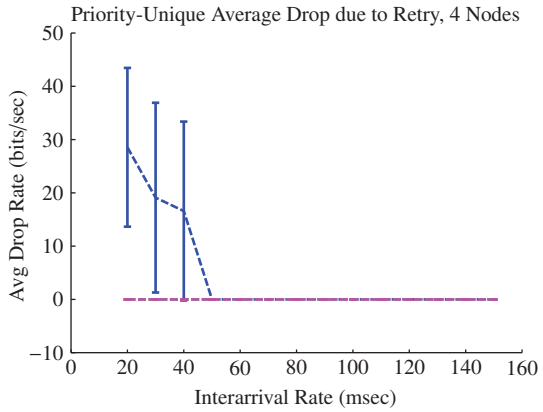


Figure A.140: N4 Retry Dropped Data

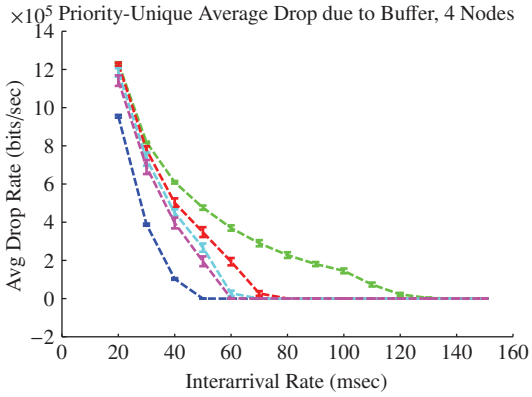


Figure A.141: N4 Buffer Dropped Data

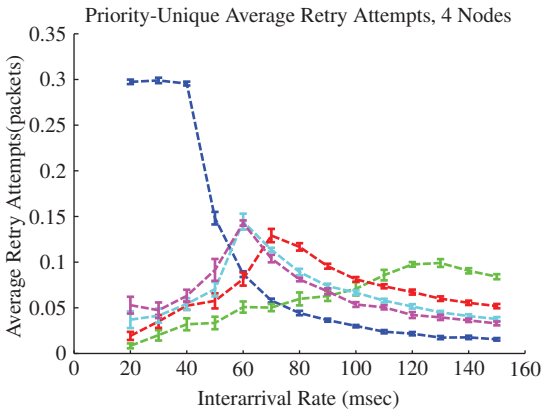


Figure A.142: N4 Retry Attempts

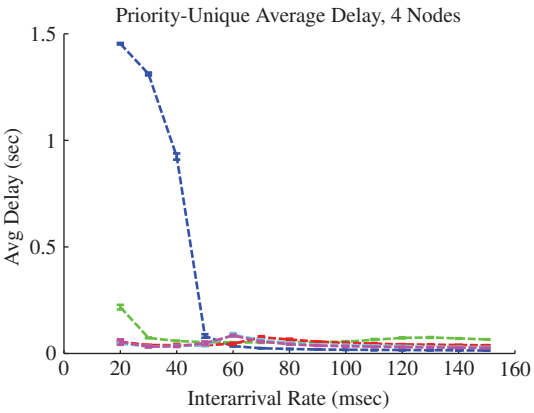


Figure A.143: N4 Packet Delay

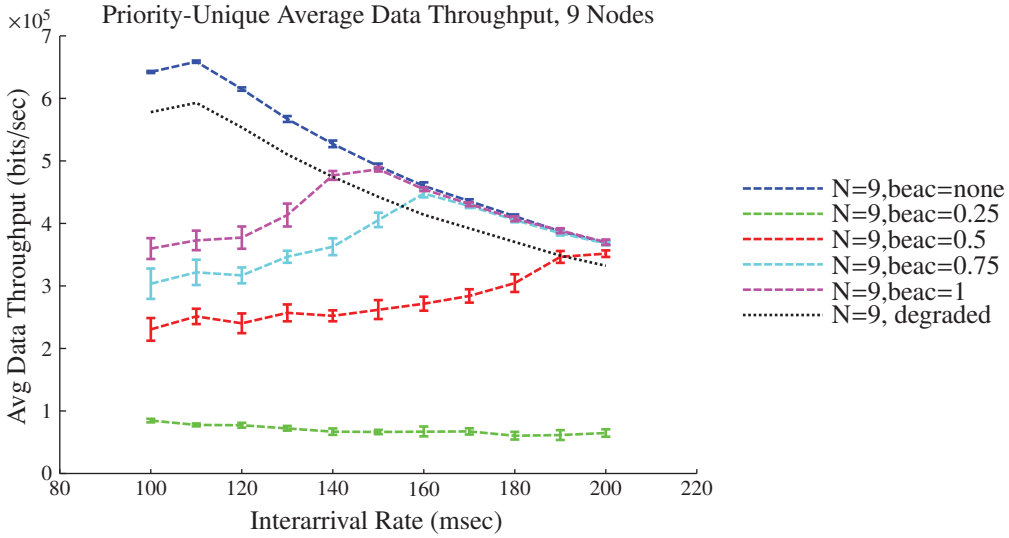


Figure A.144: N9 Data Throughput

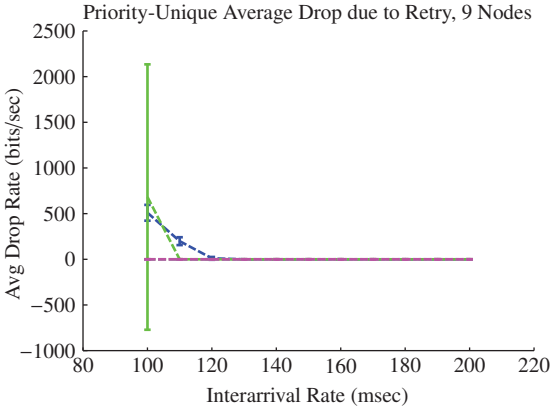


Figure A.145: N9 Retry Dropped Data

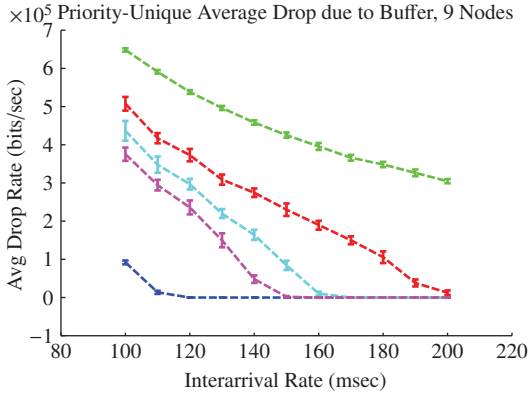


Figure A.146: N9 Buffer Dropped Data

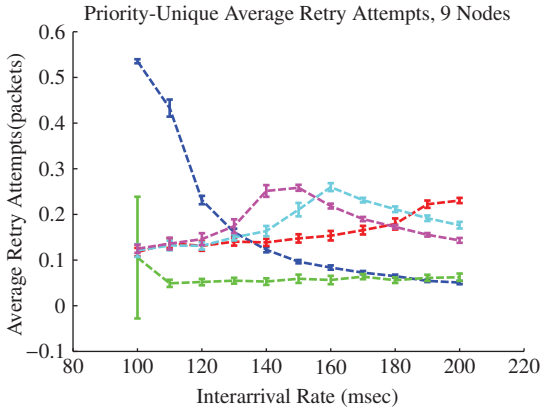


Figure A.147: N9 Retry Attempts

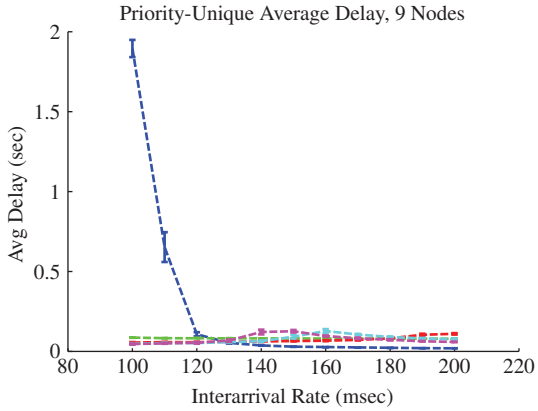


Figure A.148: N9 Packet Delay

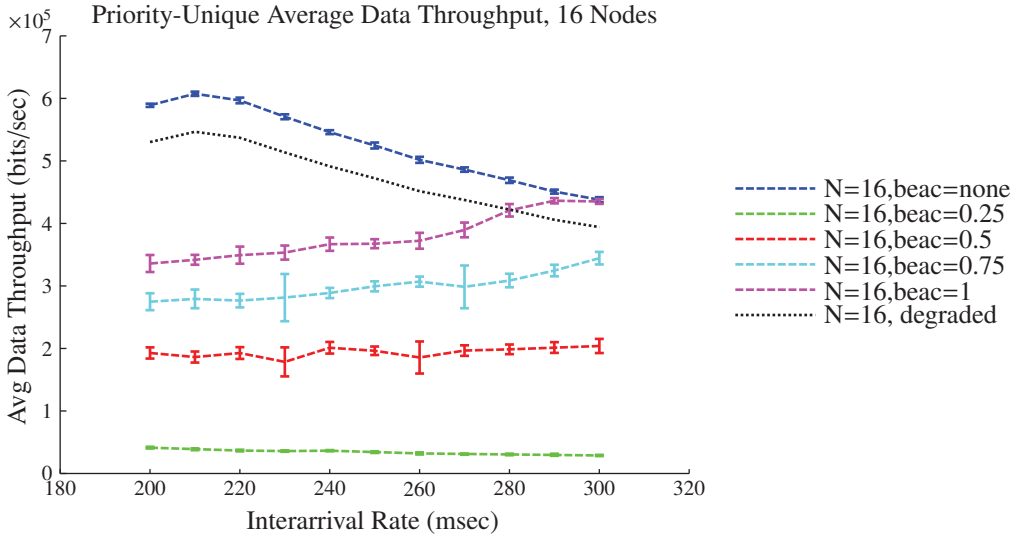


Figure A.149: N16 Data Throughput

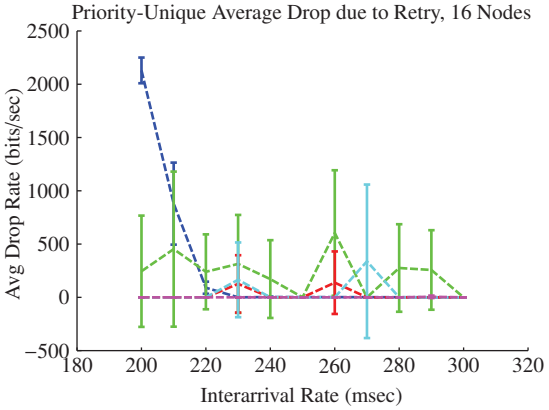


Figure A.150: N16 Retry Dropped Data

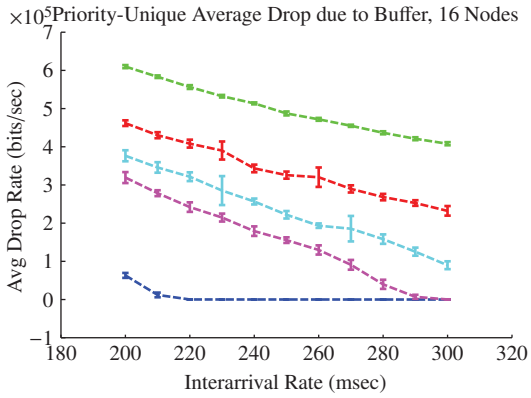


Figure A.151: N16 Buffer Dropped Data

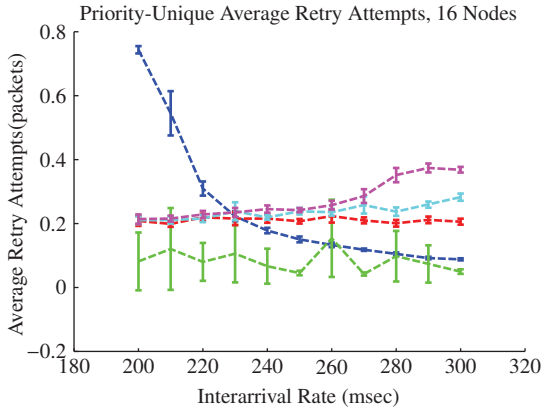


Figure A.152: N16 Retry Attempts

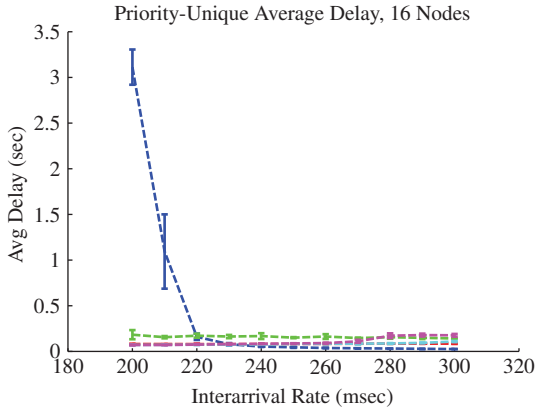


Figure A.153: N16 Packet Delay

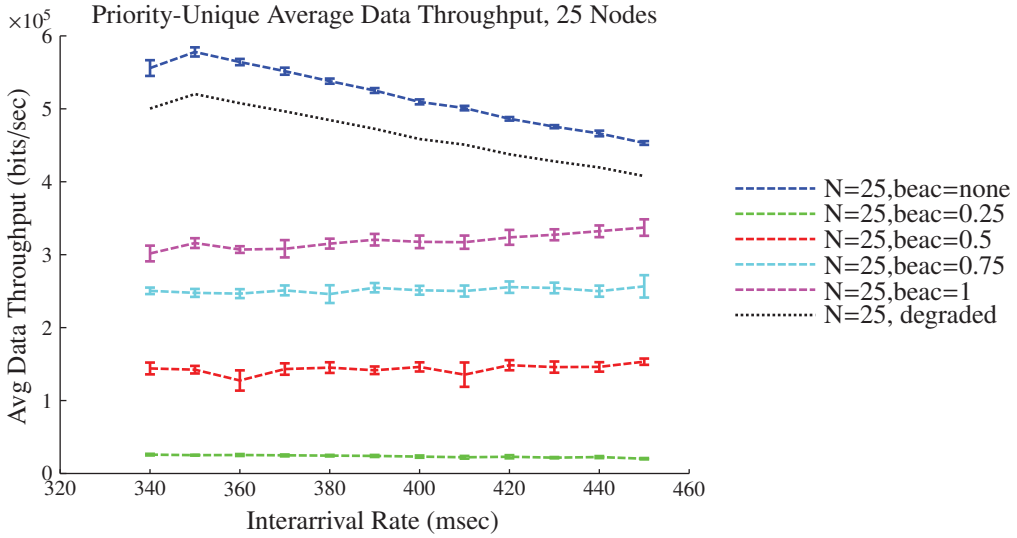


Figure A.154: N25 Data Throughput

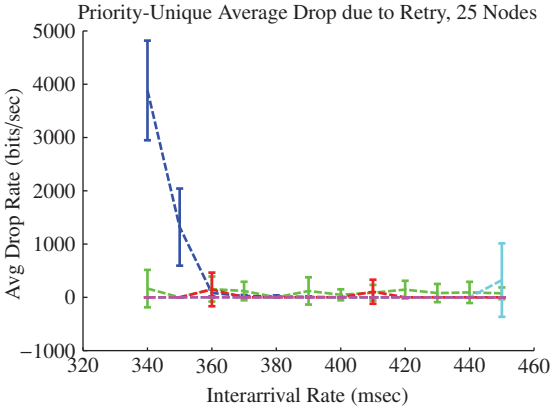


Figure A.155: N25 Retry Dropped Data

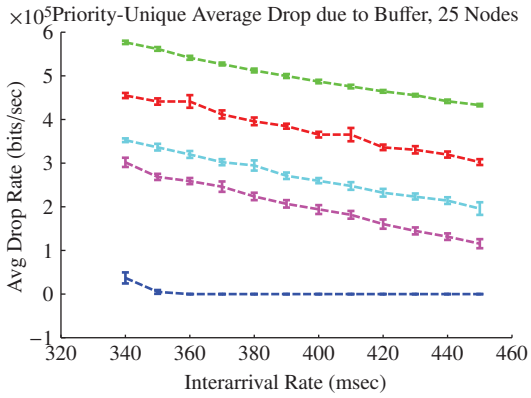


Figure A.156: N25 Buffer Dropped Data

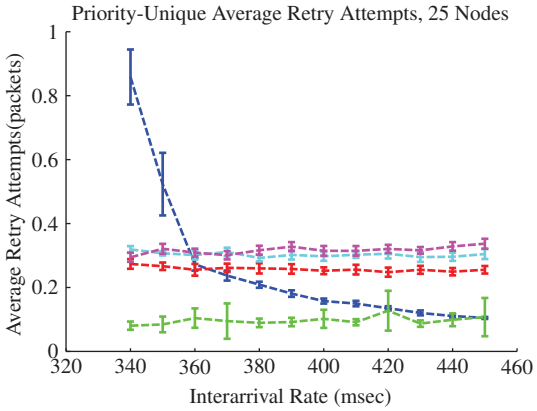


Figure A.157: N25 Retry Attempts

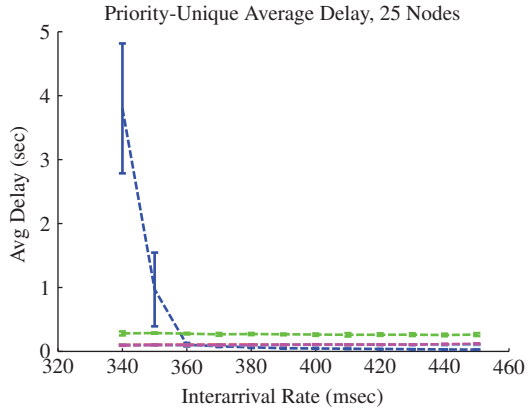


Figure A.158: N25 Packet Delay

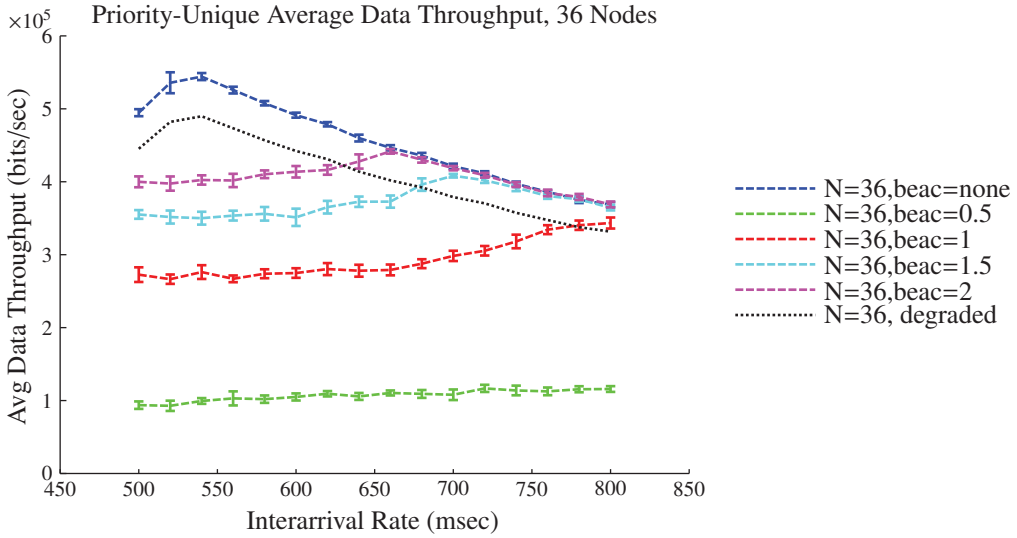


Figure A.159: N36 Data Throughput

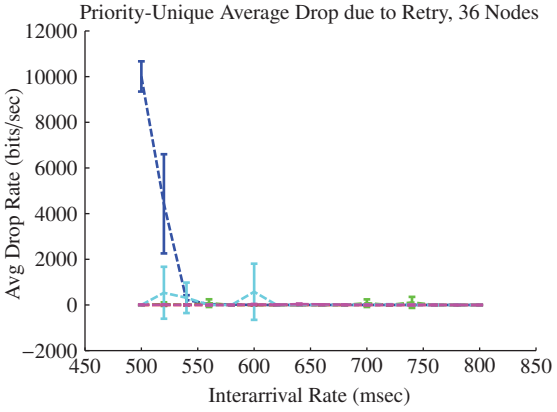


Figure A.160: N36 Retry Dropped Data

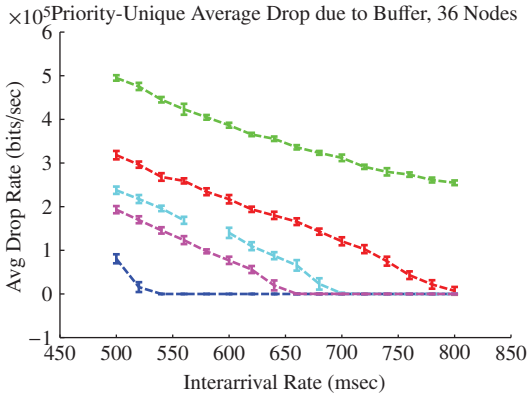


Figure A.161: N36 Buffer Dropped Data

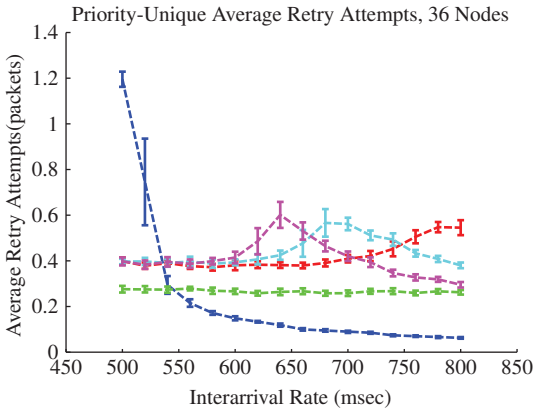


Figure A.162: N36 Retry Attempts

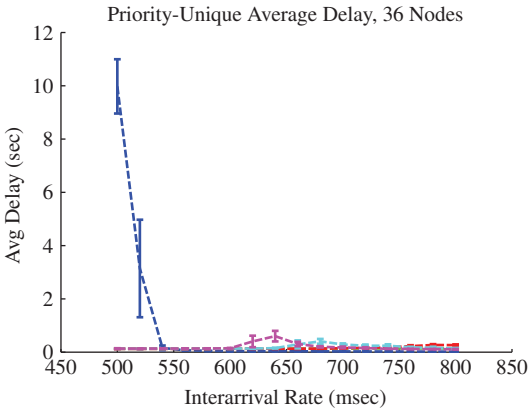


Figure A.163: N36 Packet Delay

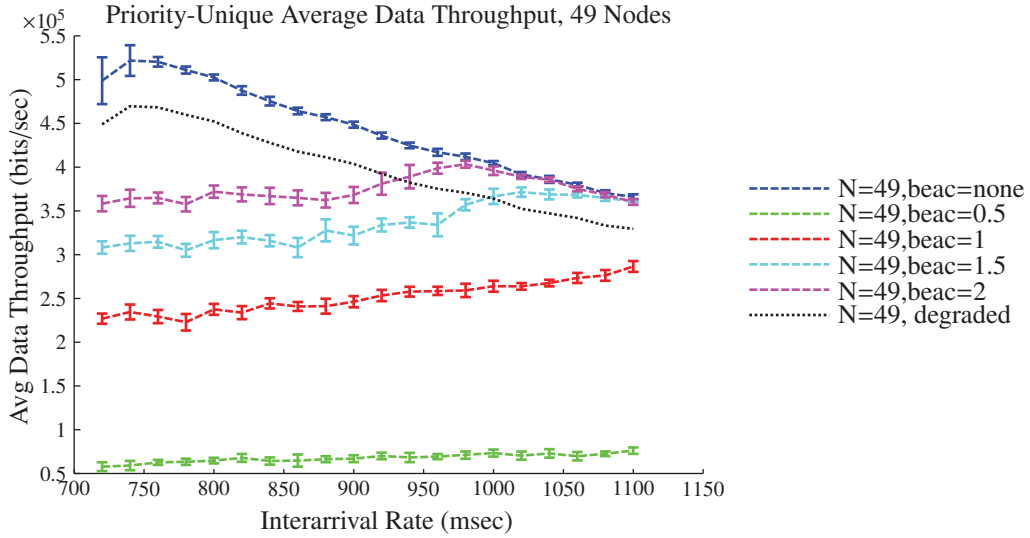


Figure A.164: N49 Data Throughput

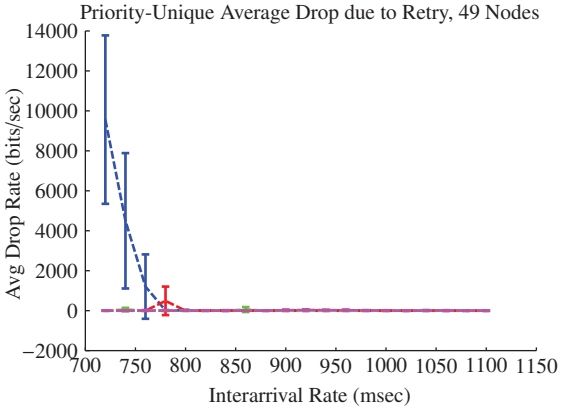


Figure A.165: N49 Retry Dropped Data

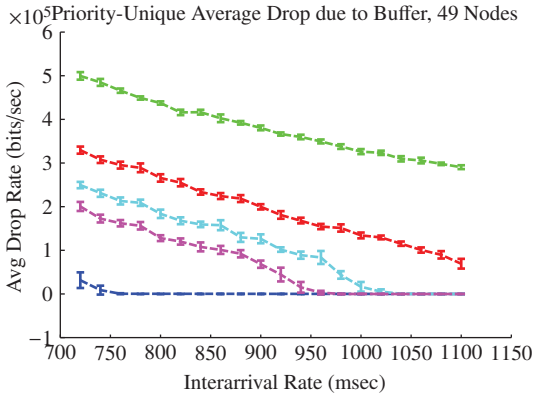


Figure A.166: N49 Buffer Dropped Data

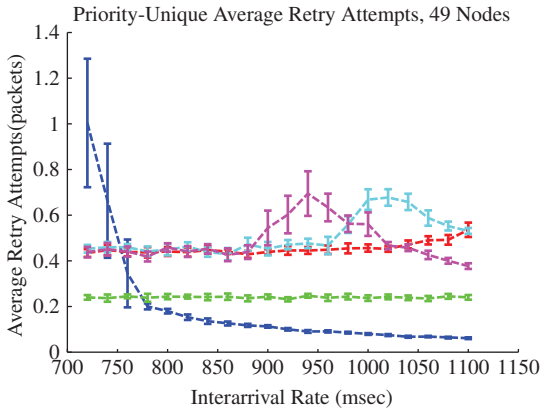


Figure A.167: N49 Retry Attempts

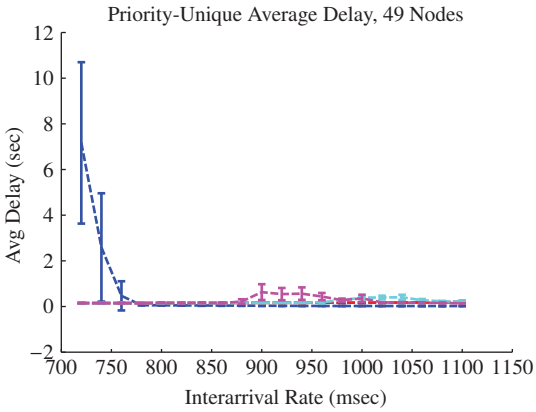


Figure A.168: N49 Packet Delay

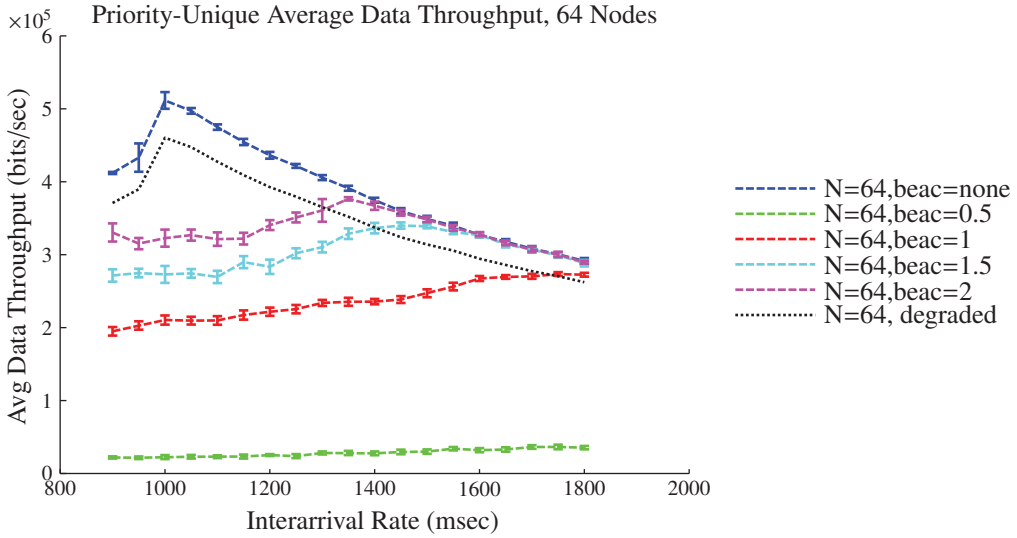


Figure A.169: N64 Data Throughput

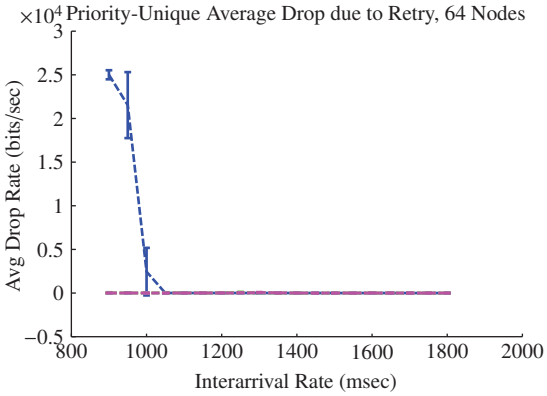


Figure A.170: N64 Retry Dropped Data

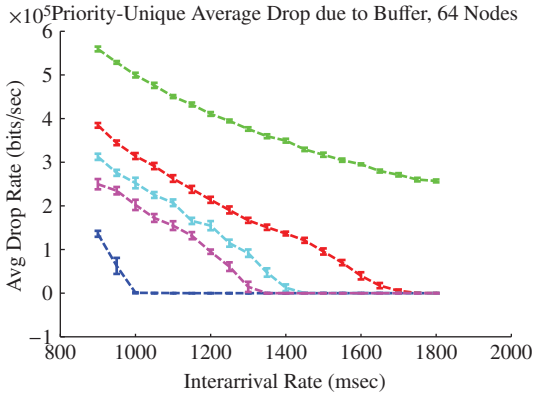


Figure A.171: N64 Buffer Dropped Data

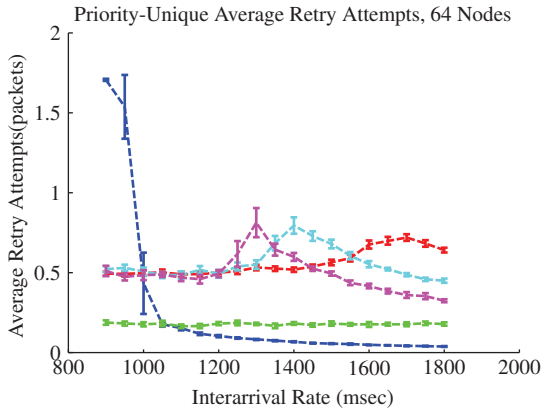


Figure A.172: N64 Retry Attempts

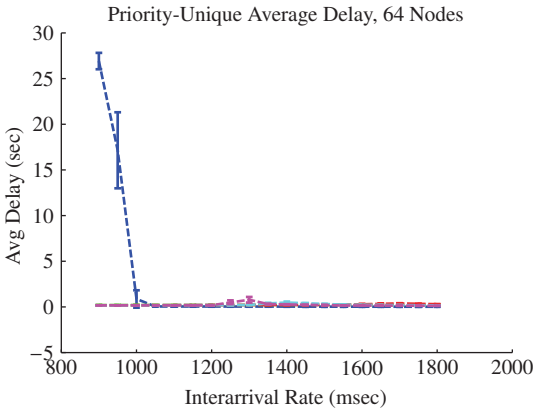


Figure A.173: N64 Packet Delay

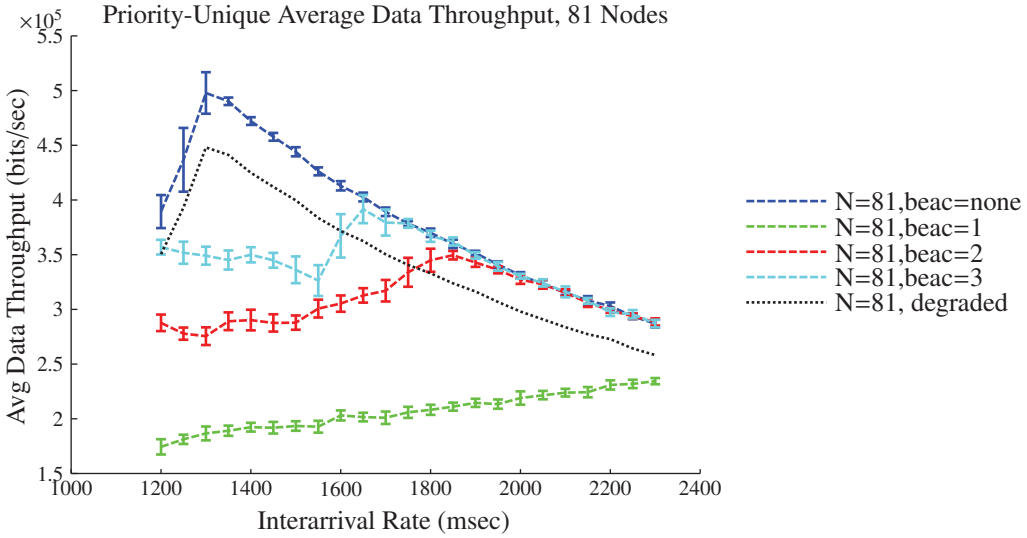


Figure A.174: N81 Data Throughput

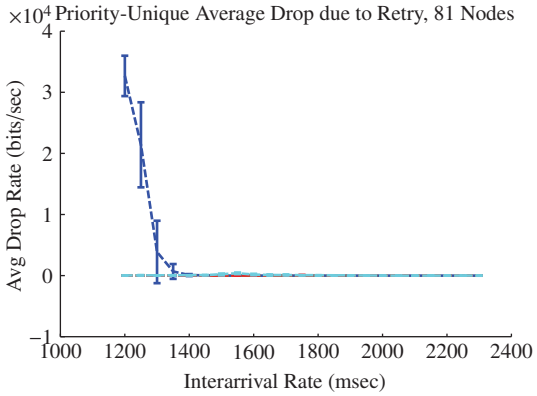


Figure A.175: N81 Retry Dropped Data

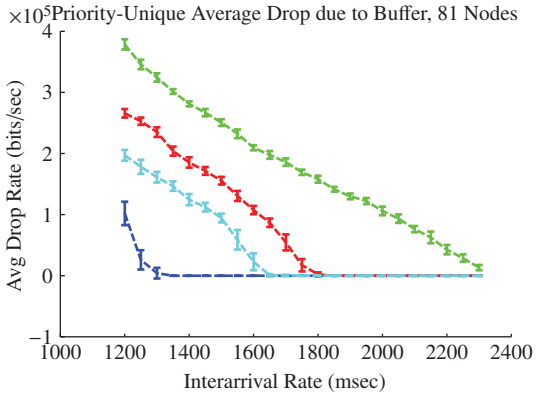


Figure A.176: N81 Buffer Dropped Data

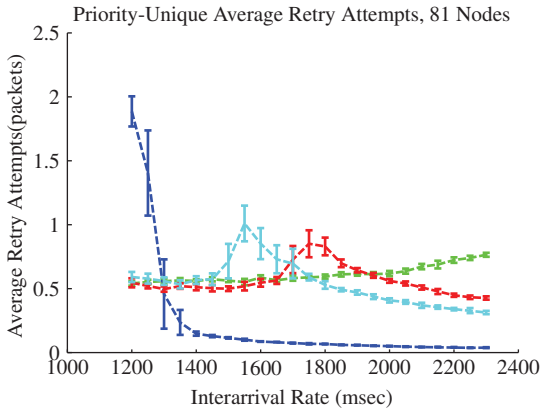


Figure A.177: N81 Retry Attempts

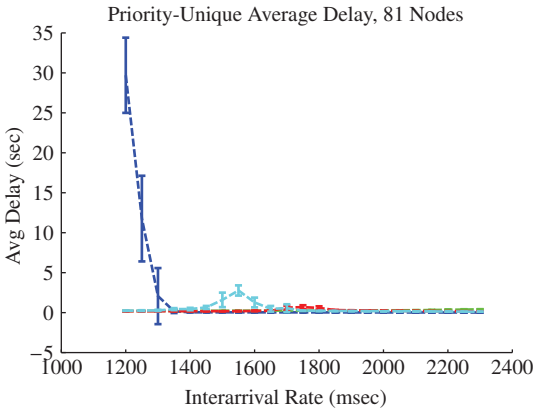


Figure A.178: N81 Packet Delay

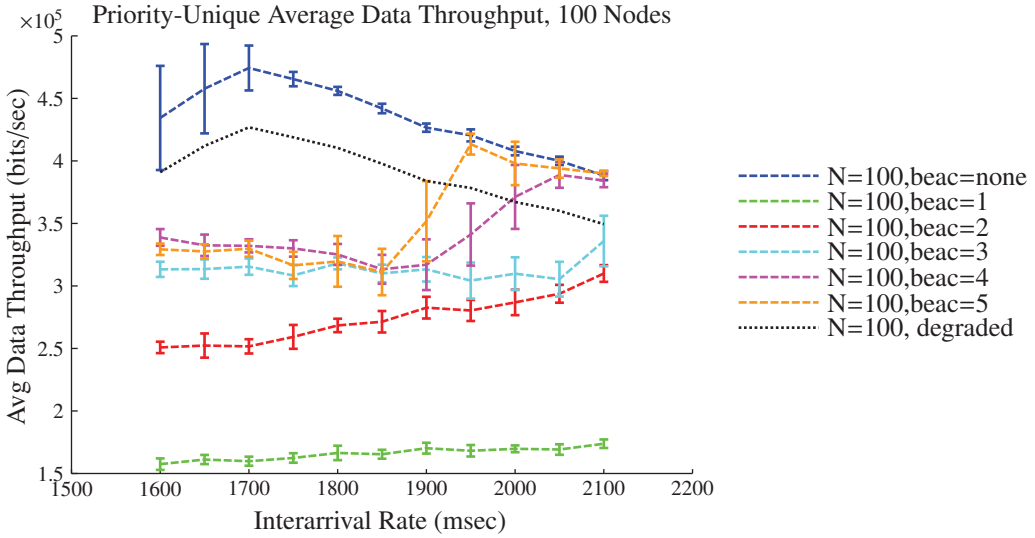


Figure A.179: N100 Data Throughput

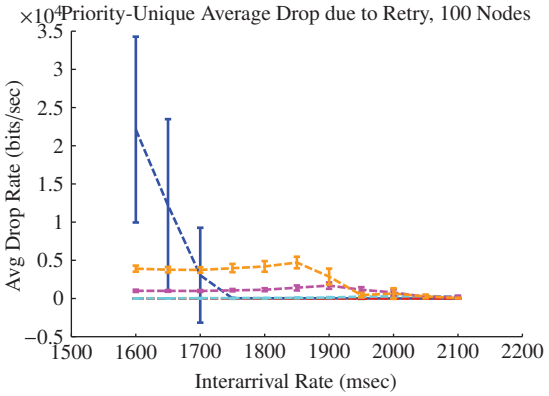


Figure A.180: N100 Retry Dropped Data

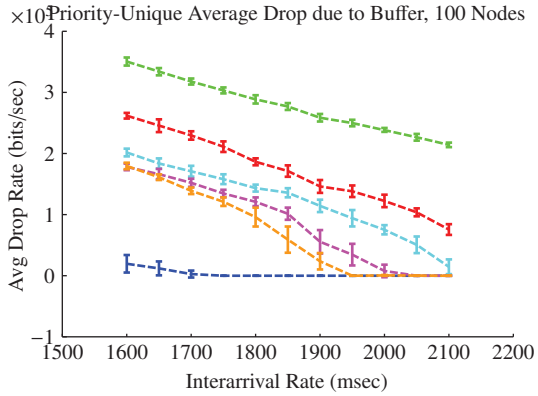


Figure A.181: N100 Buffer Dropped Data

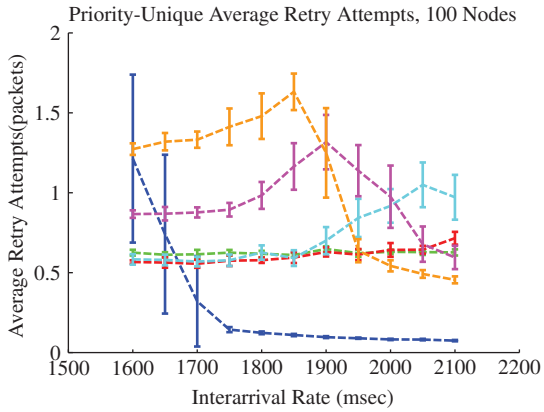


Figure A.182: N100 Retry Attempts

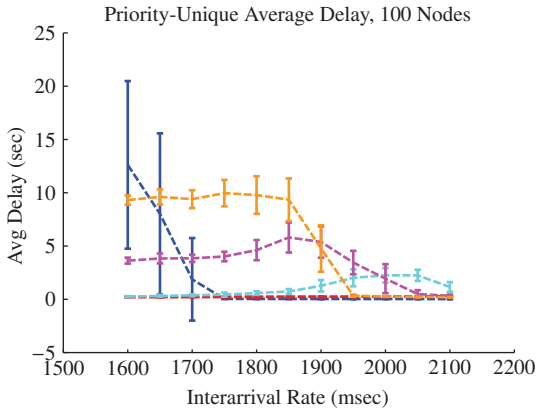


Figure A.183: N100 Packet Delay

A.5.2 Identical Scenario.

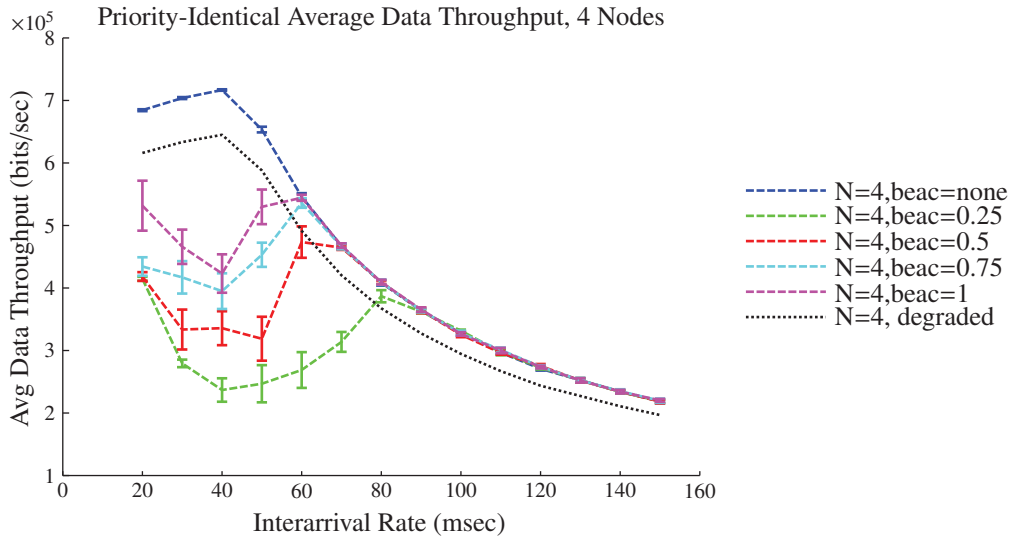


Figure A.184: N4 Data Throughput

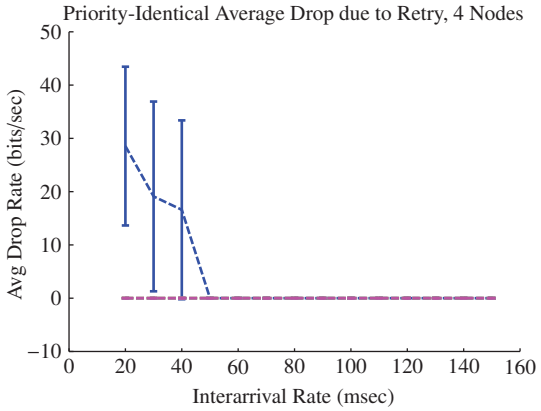


Figure A.185: N4 Retry Dropped Data

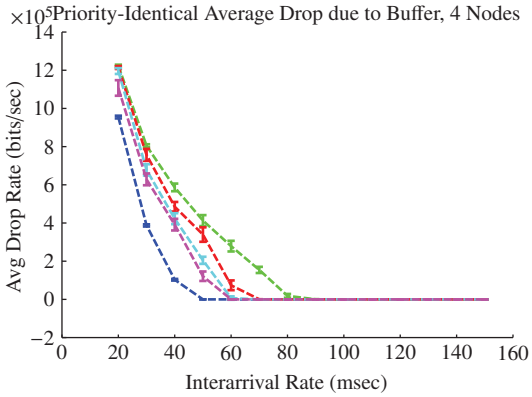


Figure A.186: N4 Buffer Dropped Data

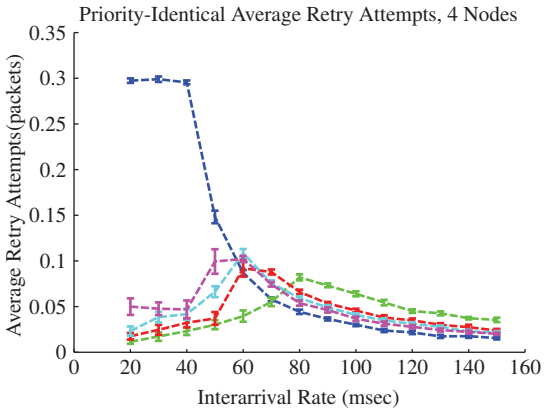


Figure A.187: N4 Retry Attempts

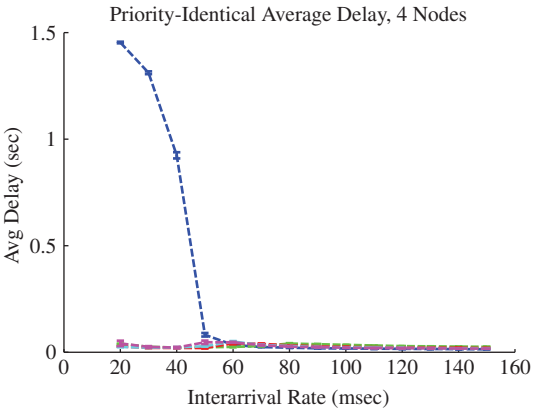


Figure A.188: N4 Packet Delay

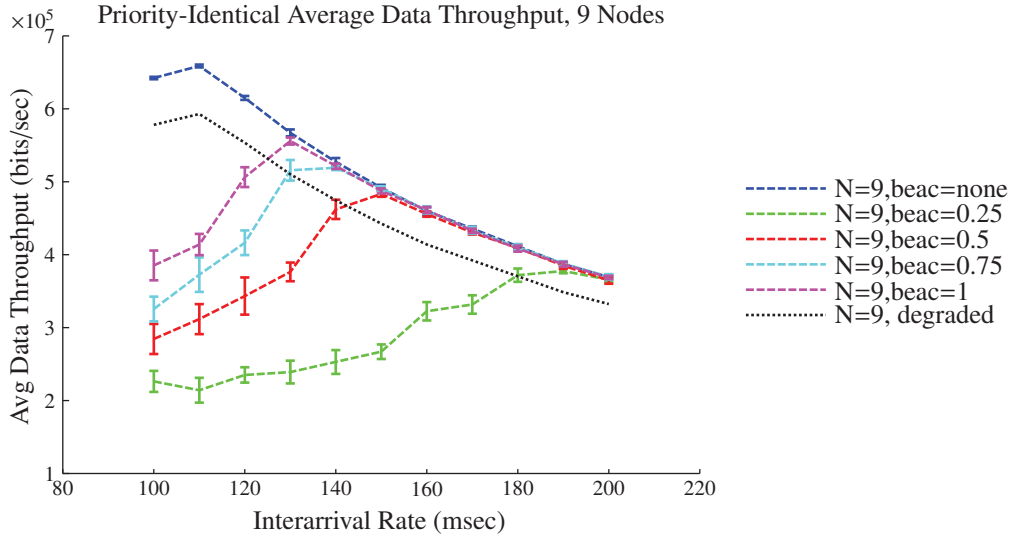


Figure A.189: N9 Data Throughput

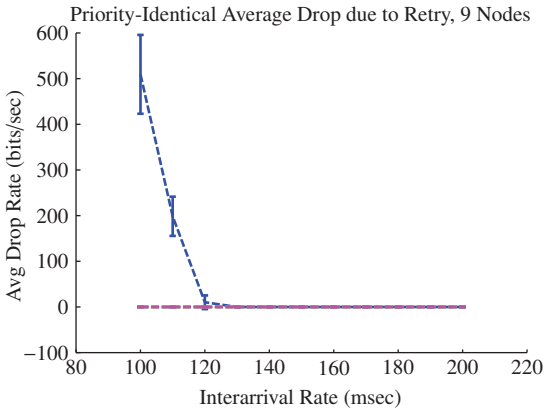


Figure A.190: N9 Retry Dropped Data

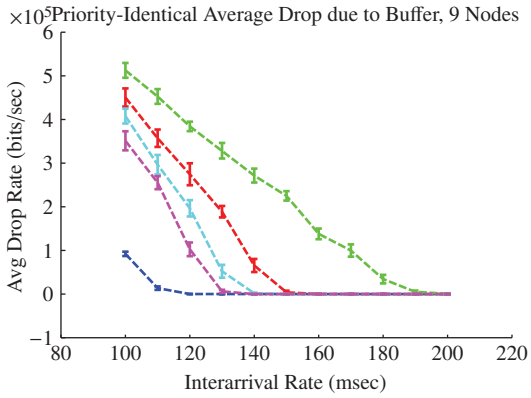


Figure A.191: N9 Buffer Dropped Data

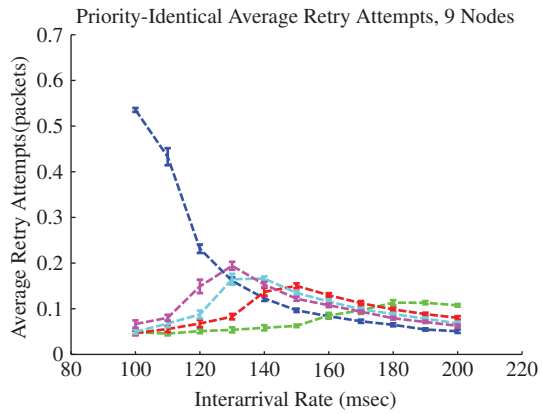


Figure A.192: N9 Retry Attempts

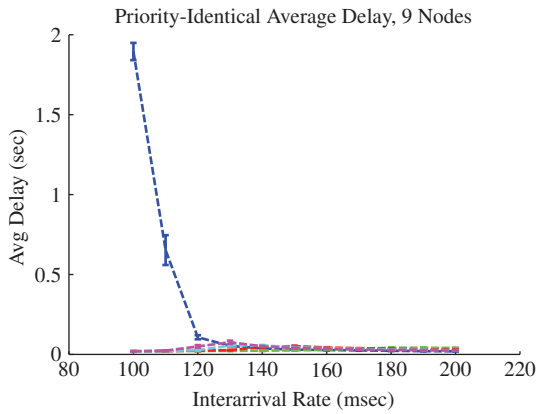


Figure A.193: N9 Packet Delay

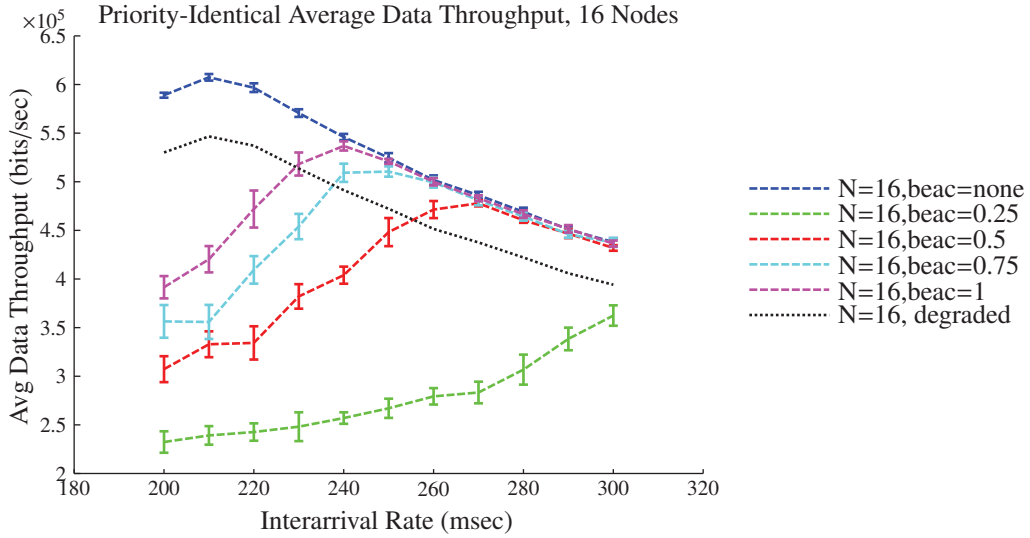


Figure A.194: N16 Data Throughput

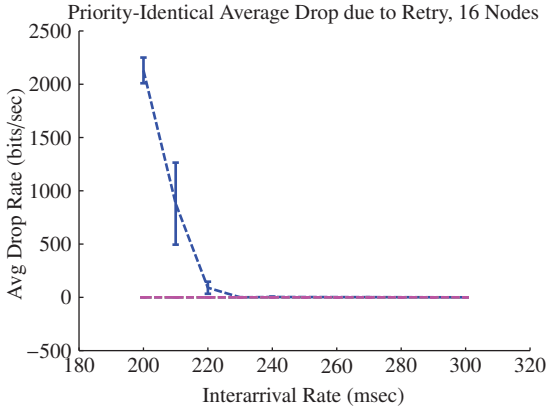


Figure A.195: N16 Retry Dropped Data

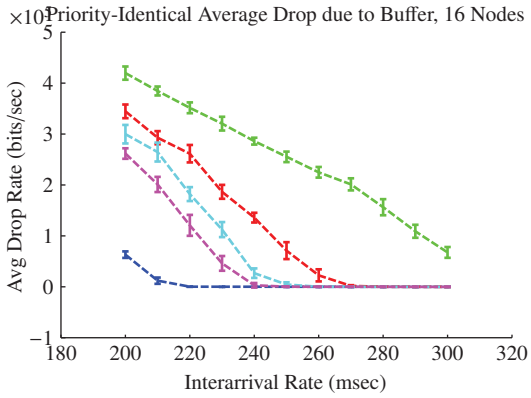


Figure A.196: N16 Buffer Dropped Data

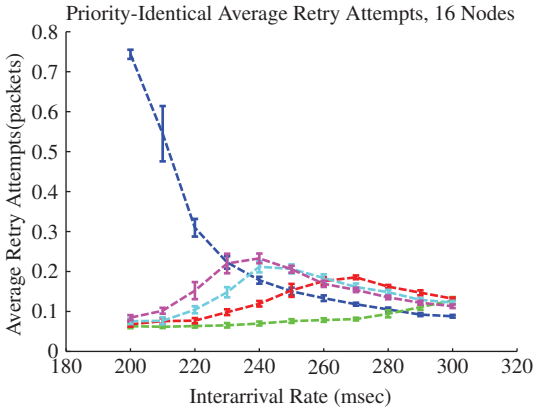


Figure A.197: N16 Retry Attempts

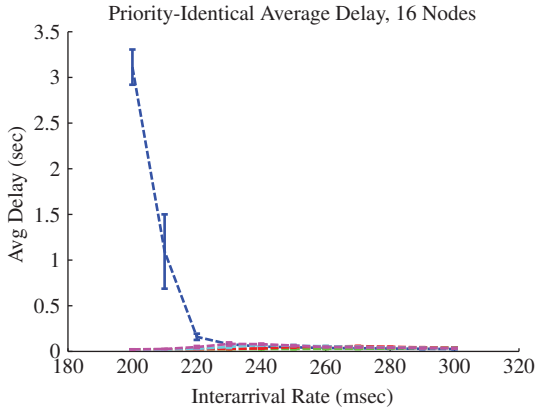


Figure A.198: N16 Packet Delay

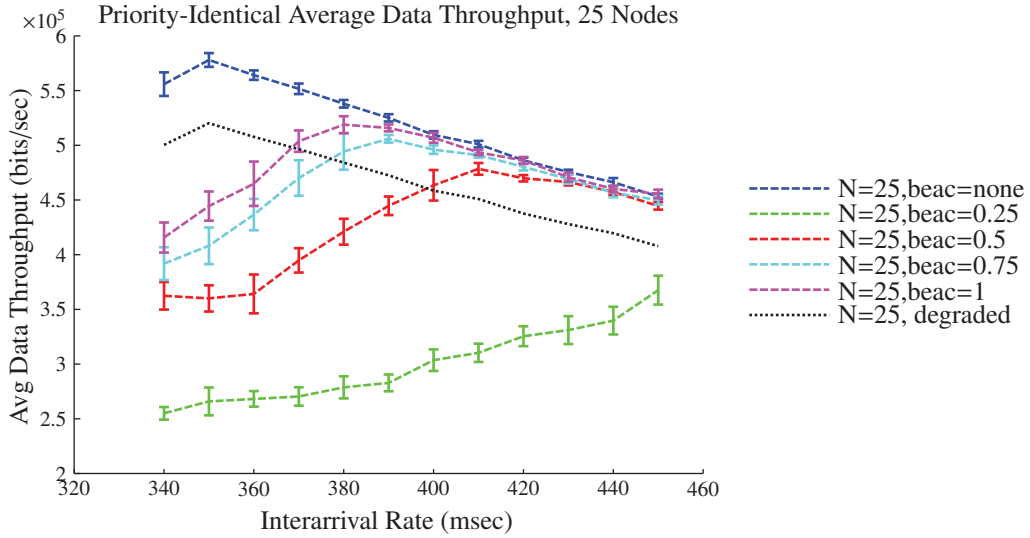


Figure A.199: N25 Data Throughput

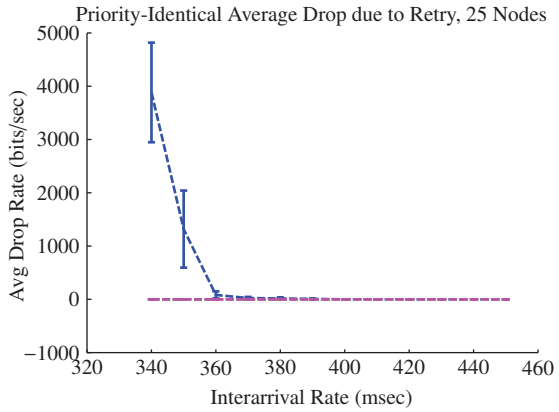


Figure A.200: N25 Retry Dropped Data

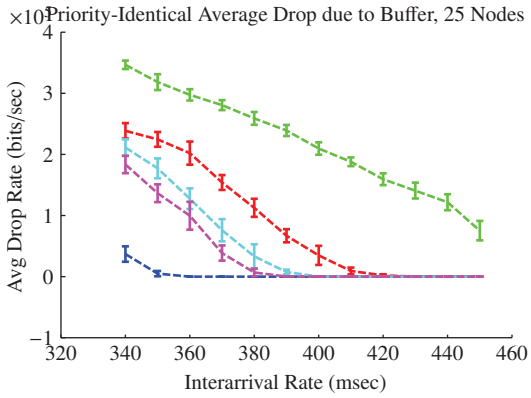


Figure A.201: N25 Buffer Dropped Data

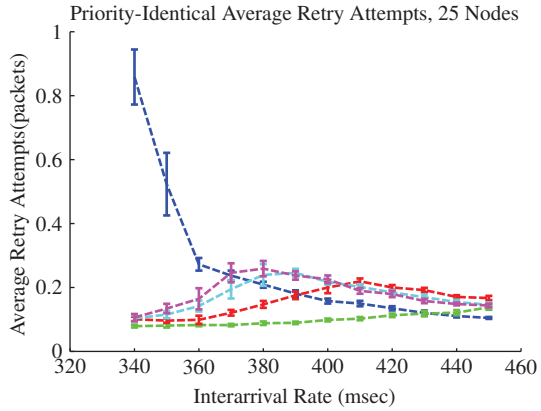


Figure A.202: N25 Retry Attempts

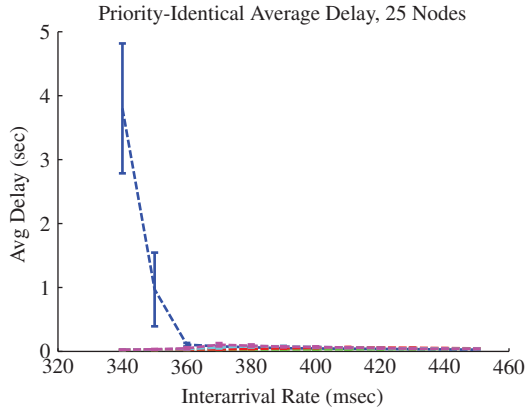


Figure A.203: N25 Packet Delay

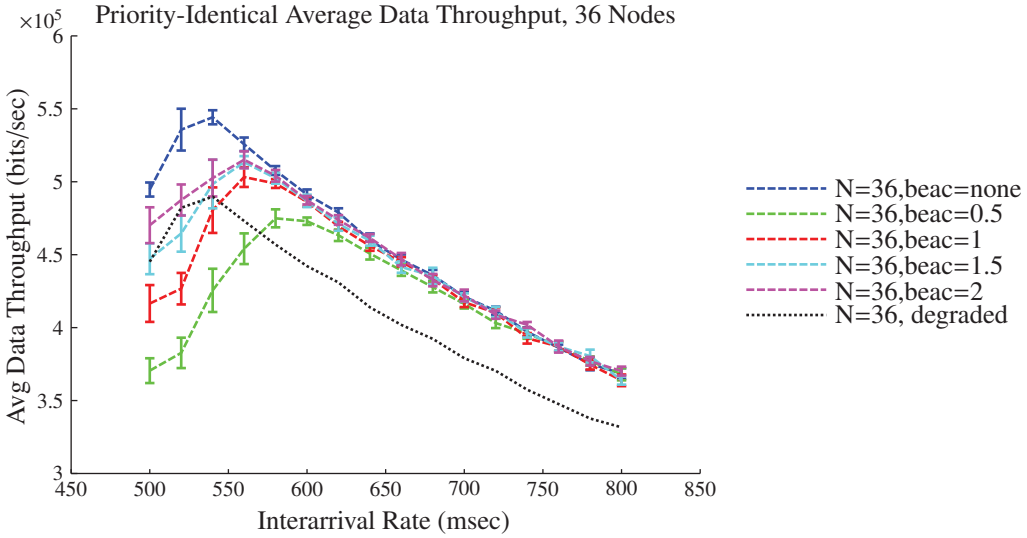


Figure A.204: N36 Data Throughput

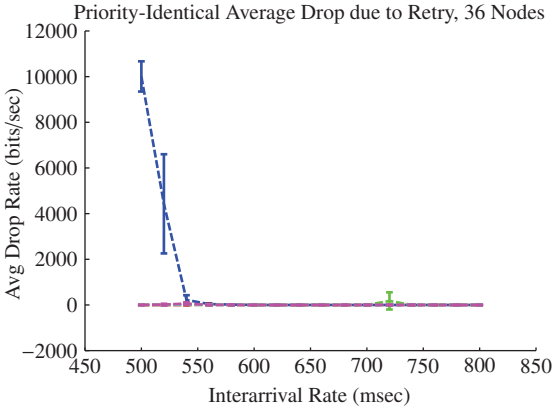


Figure A.205: N36 Retry Dropped Data

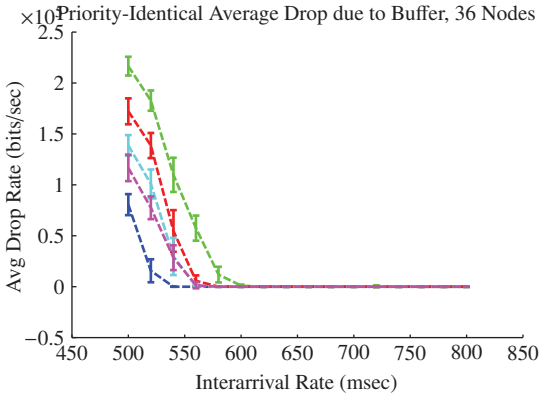


Figure A.206: N36 Buffer Dropped Data

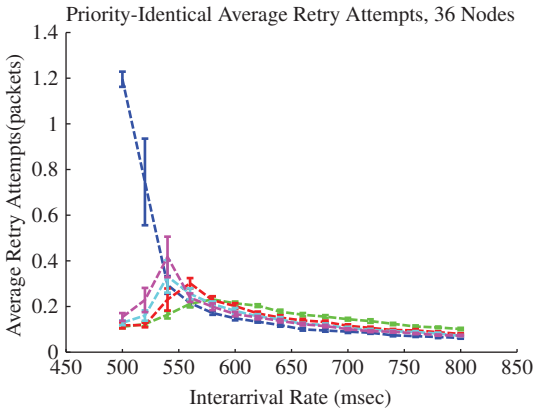


Figure A.207: N36 Retry Attempts

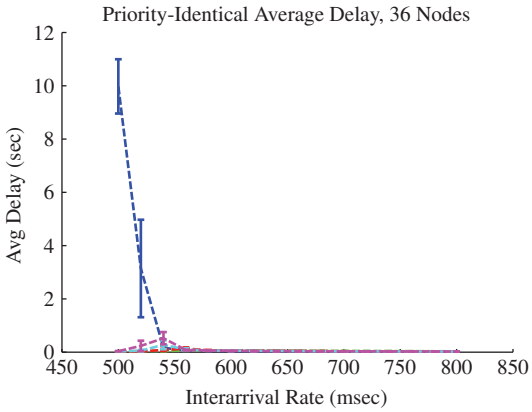


Figure A.208: N36 Packet Delay

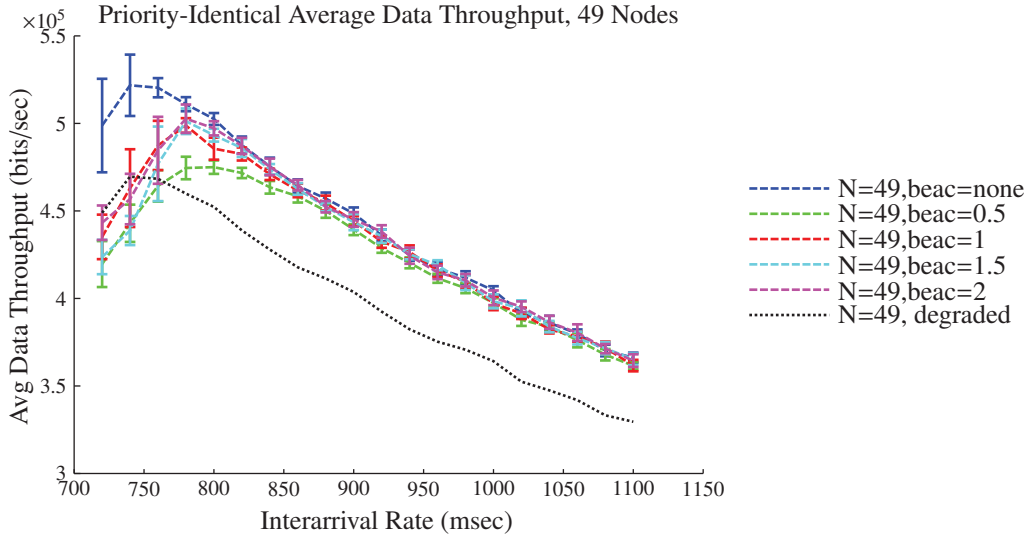


Figure A.209: N49 Data Throughput

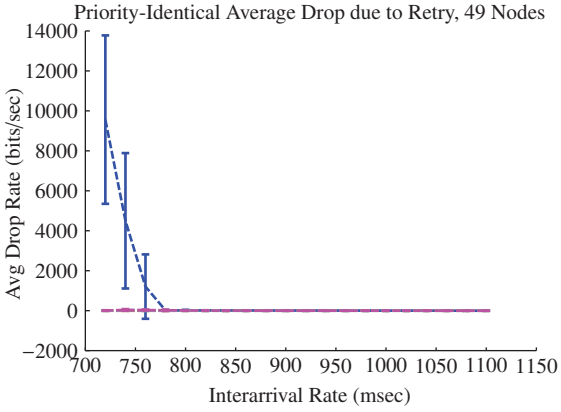


Figure A.210: N49 Retry Dropped Data

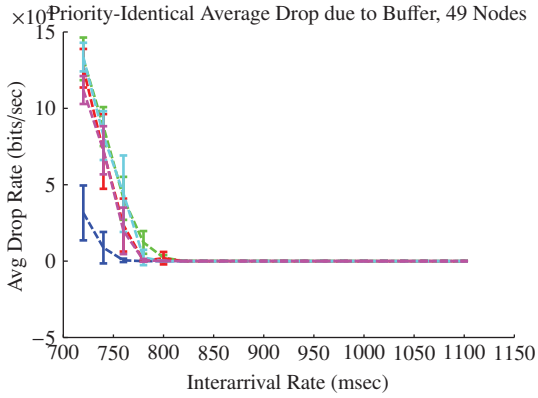


Figure A.211: N49 Buffer Dropped Data

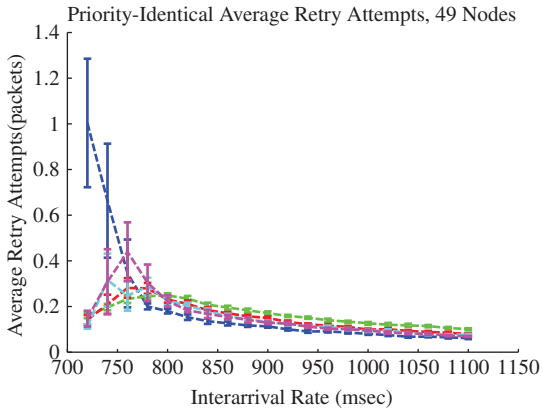


Figure A.212: N49 Retry Attempts

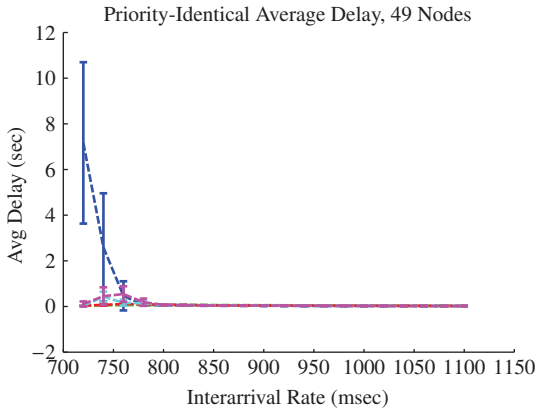


Figure A.213: N49 Packet Delay

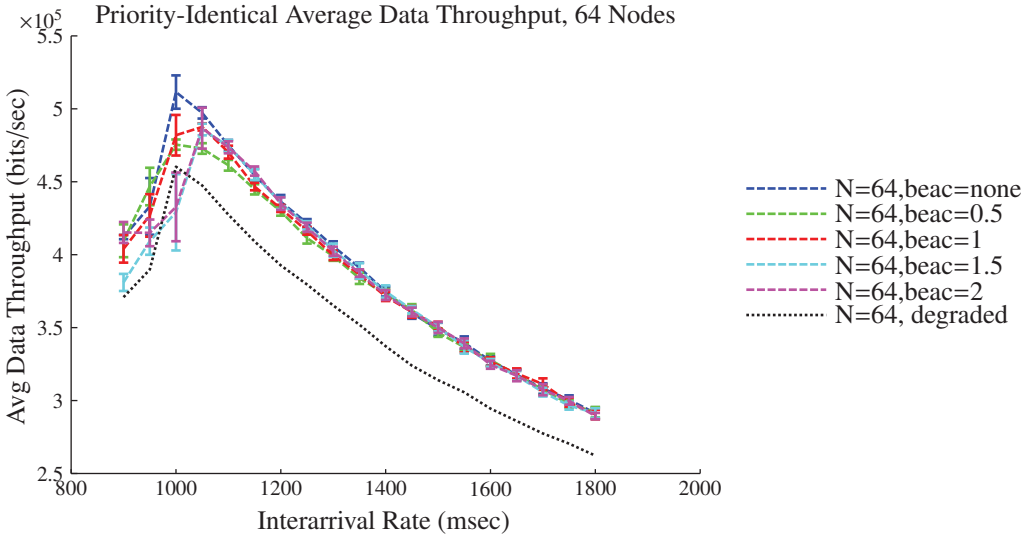


Figure A.214: N64 Data Throughput

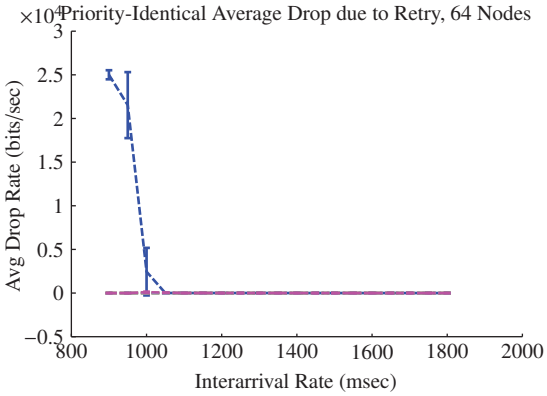


Figure A.215: N64 Retry Dropped Data

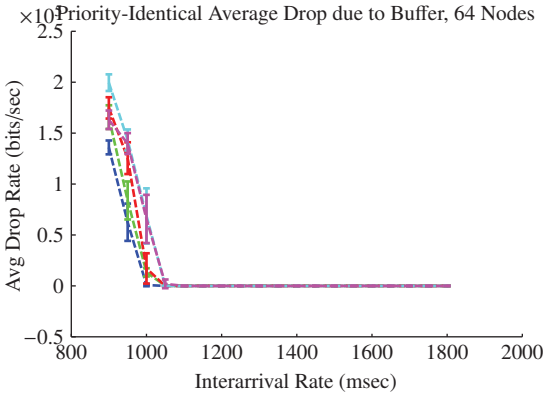


Figure A.216: N64 Buffer Dropped Data

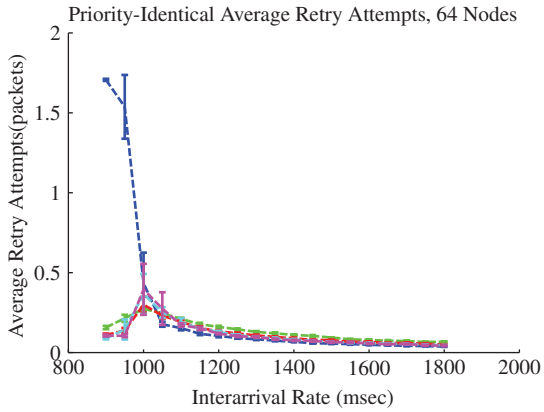


Figure A.217: N64 Retry Attempts

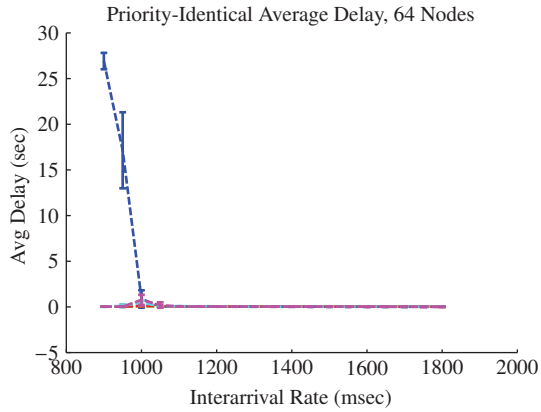


Figure A.218: N64 Packet Delay

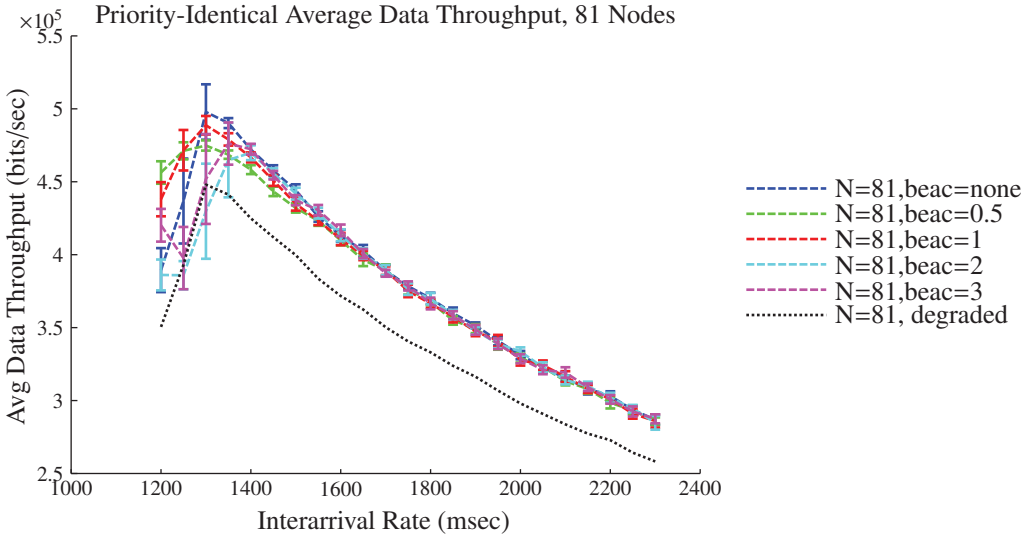


Figure A.219: N81 Data Throughput

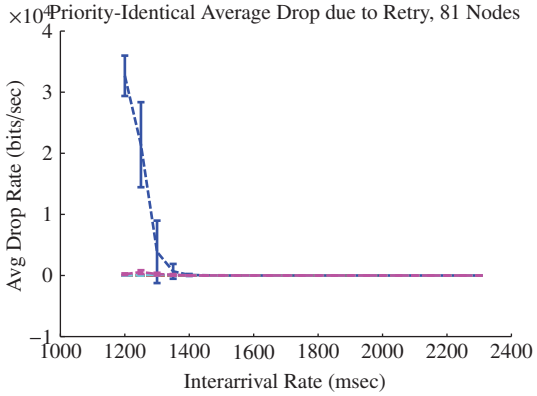


Figure A.220: N81 Retry Dropped Data

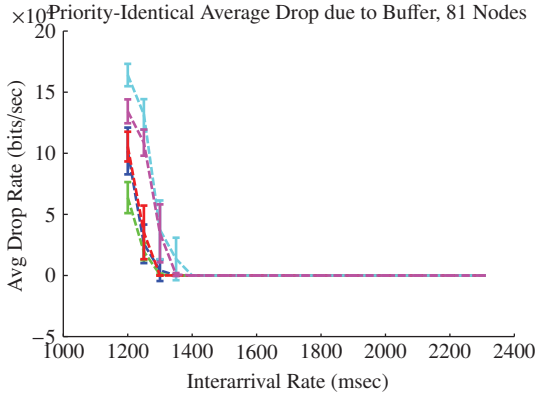


Figure A.221: N81 Buffer Dropped Data

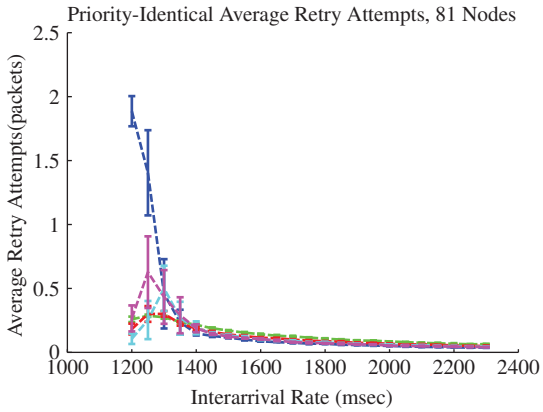


Figure A.222: N81 Retry Attempts

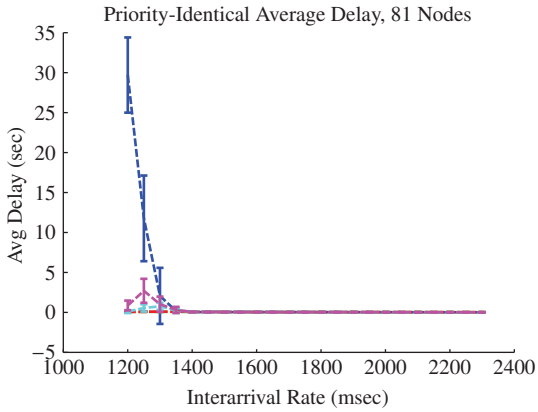


Figure A.223: N81 Packet Delay

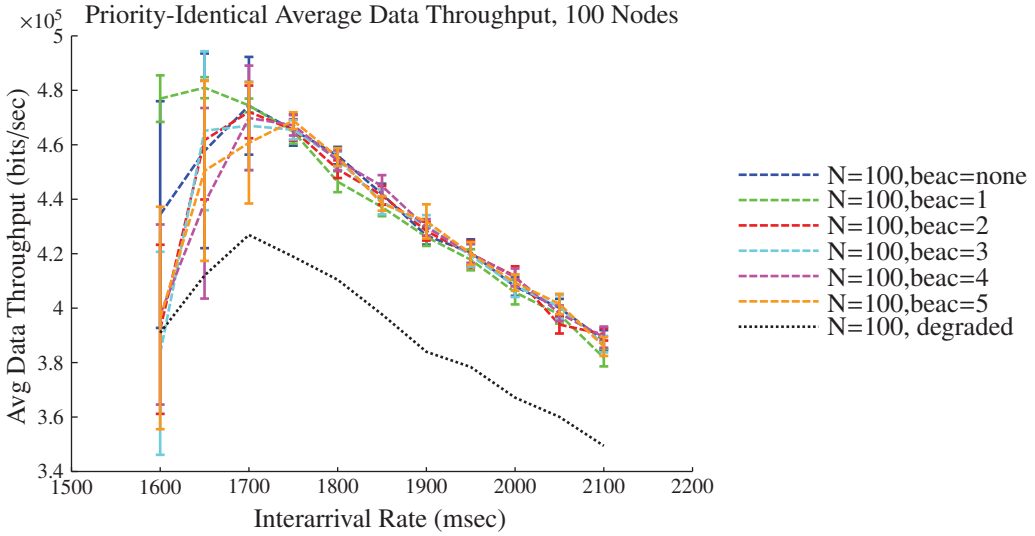


Figure A.224: N100 Data Throughput

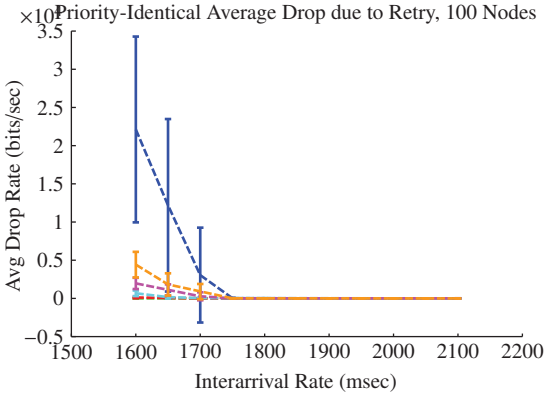


Figure A.225: N100 Retry Dropped Data

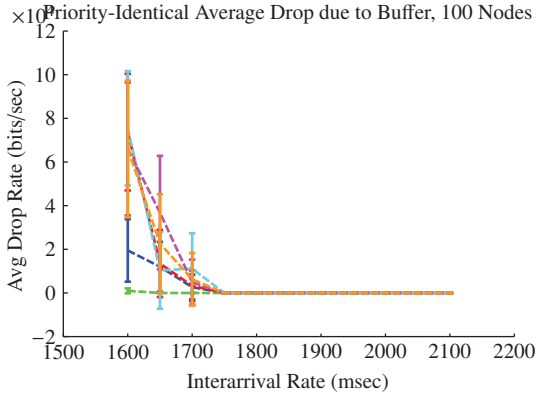


Figure A.226: N100 Buffer Dropped Data

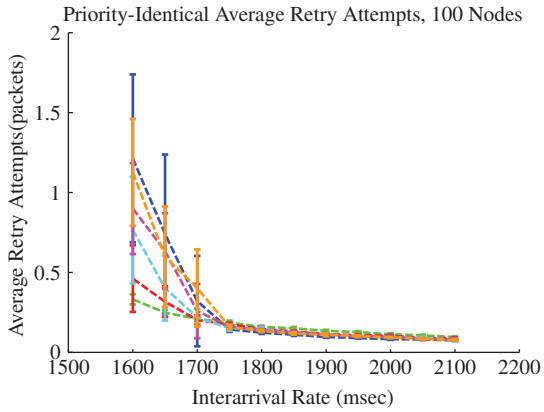


Figure A.227: N100 Retry Attempts

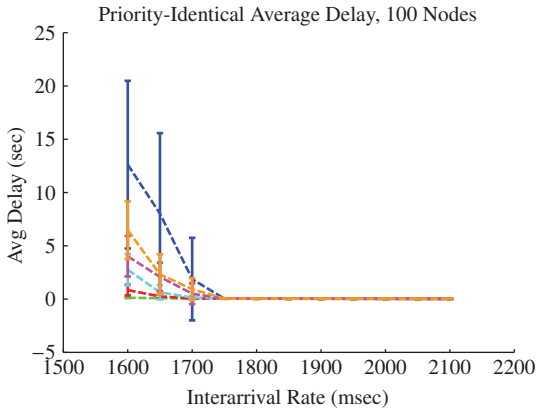


Figure A.228: N100 Packet Delay

A.5.3 Random Scenario.

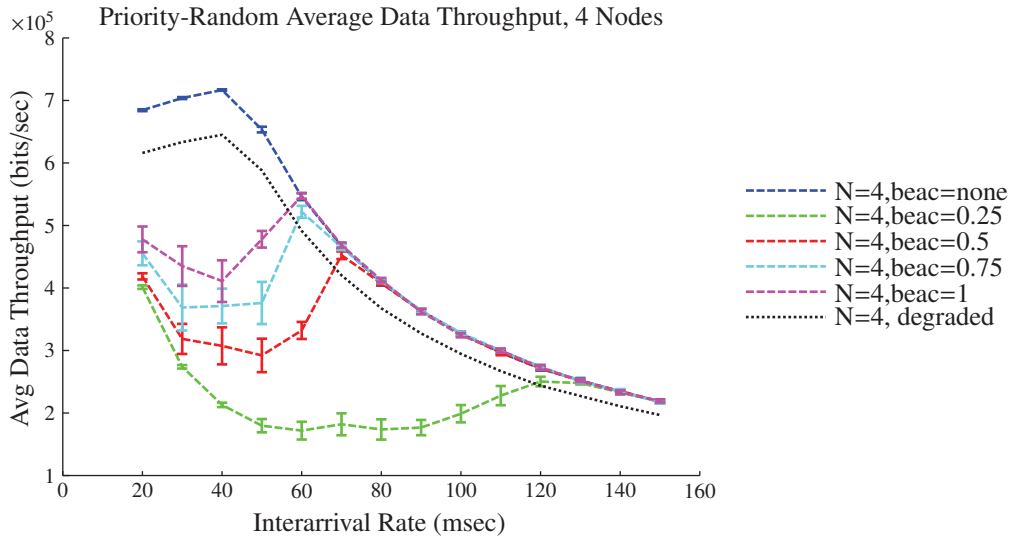


Figure A.229: N4 Data Throughput

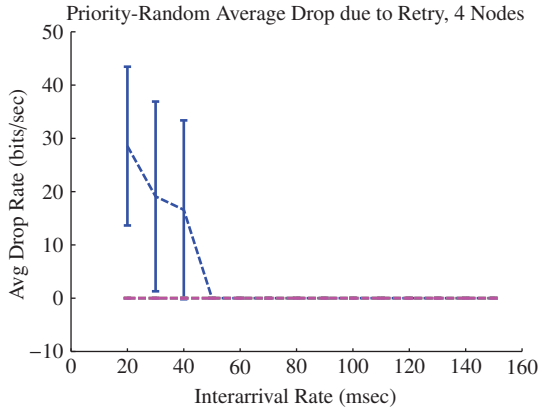


Figure A.230: N4 Retry Dropped Data

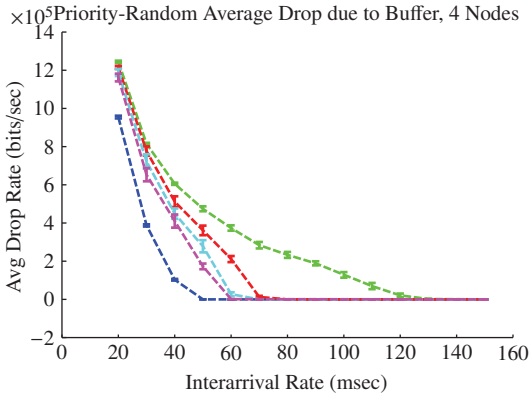


Figure A.231: N4 Buffer Dropped Data

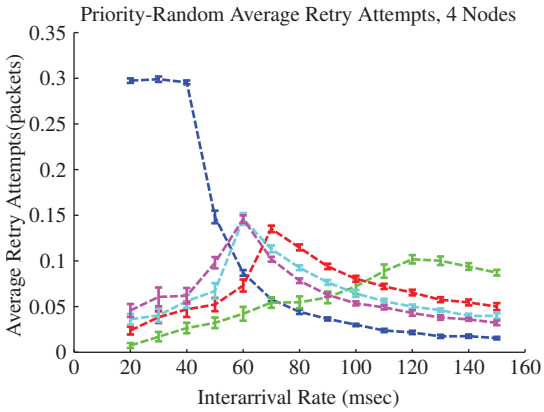


Figure A.232: N4 Retry Attempts

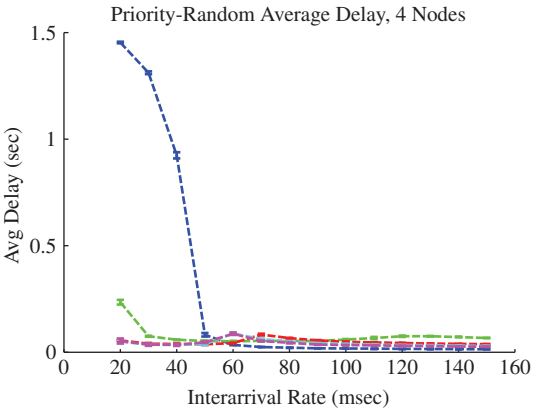


Figure A.233: N4 Packet Delay

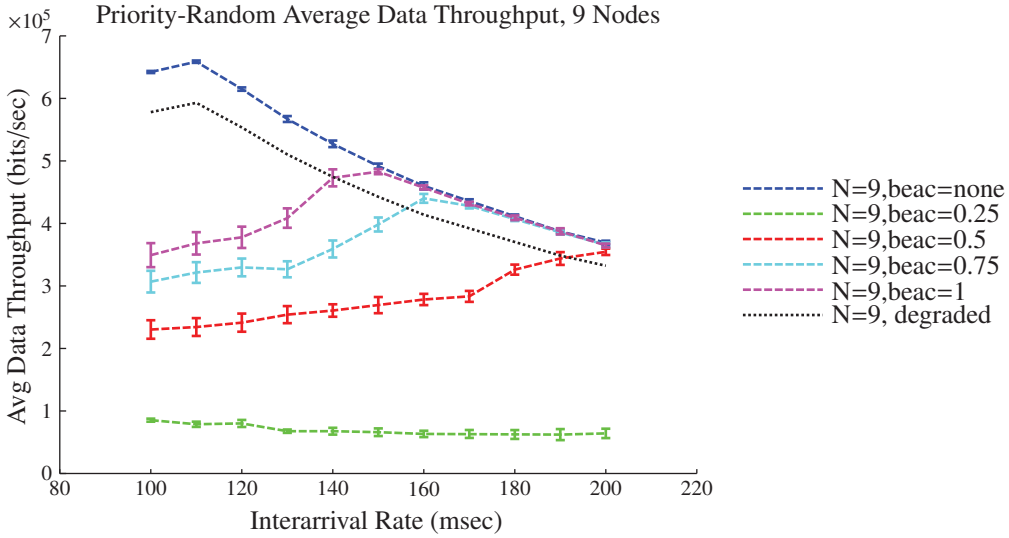


Figure A.234: N9 Data Throughput

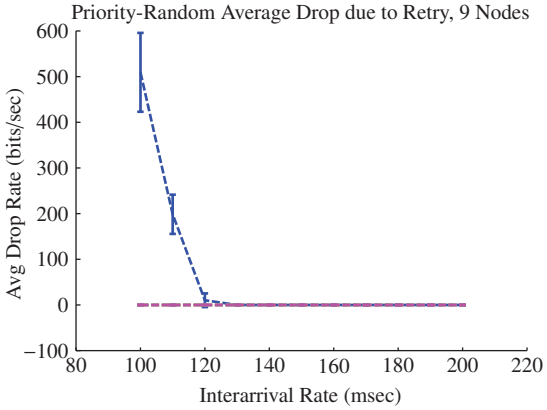


Figure A.235: N9 Retry Dropped Data

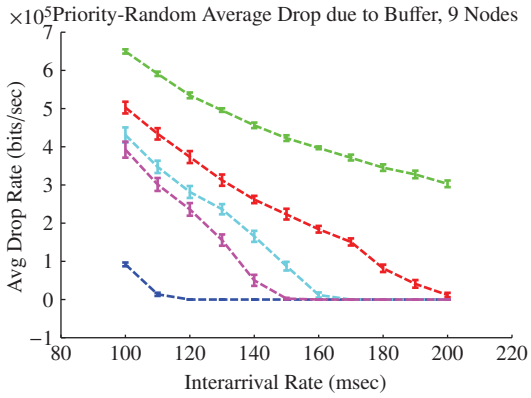


Figure A.236: N9 Buffer Dropped Data

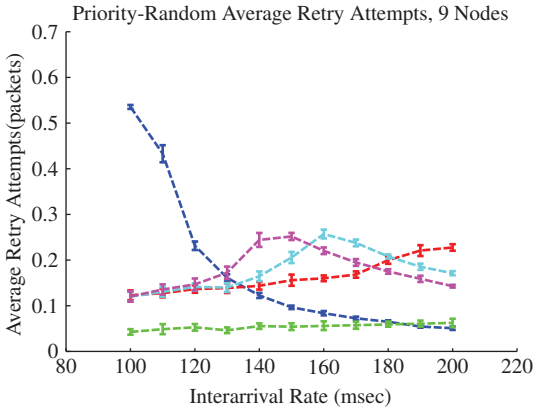


Figure A.237: N9 Retry Attempts

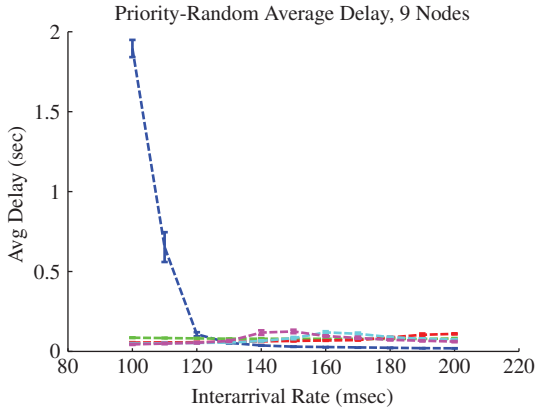


Figure A.238: N9 Packet Delay

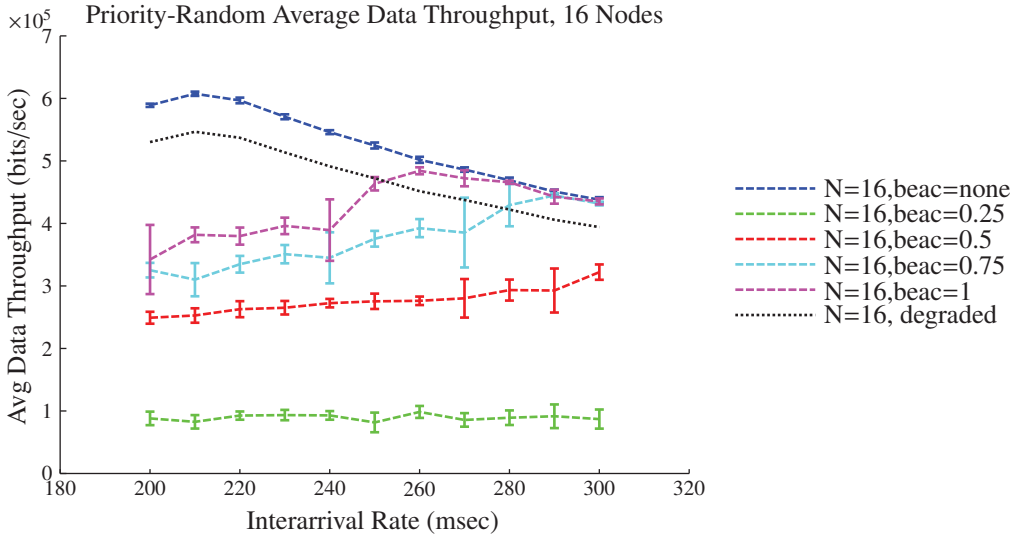


Figure A.239: N16 Data Throughput

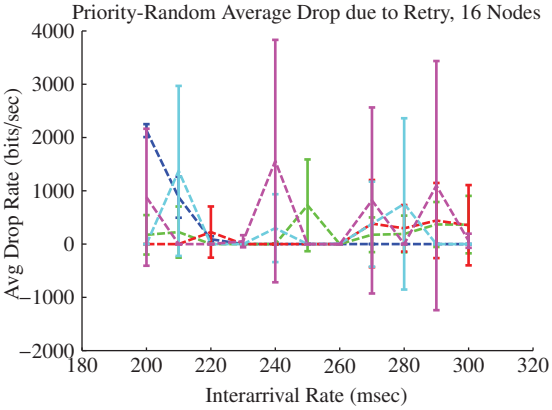


Figure A.240: N16 Retry Dropped Data

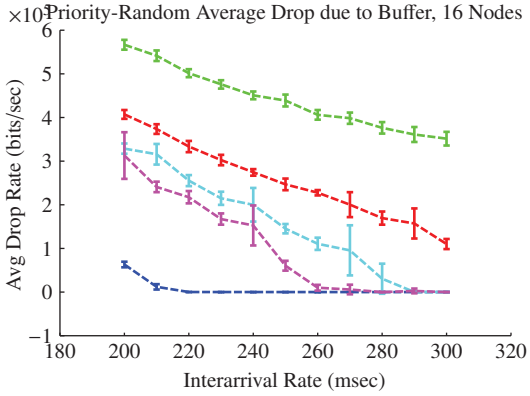


Figure A.241: N16 Buffer Dropped Data

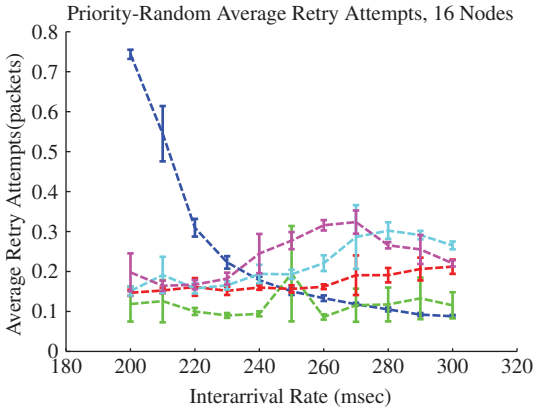


Figure A.242: N16 Retry Attempts

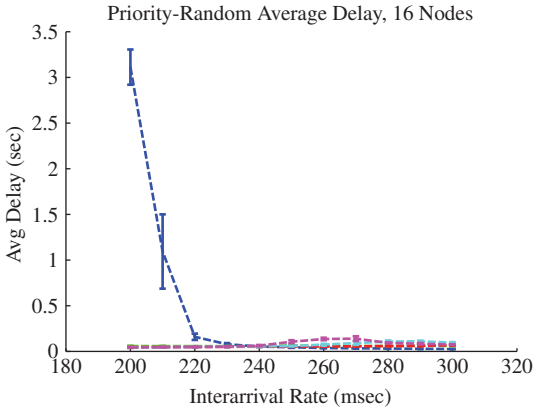


Figure A.243: N16 Packet Delay

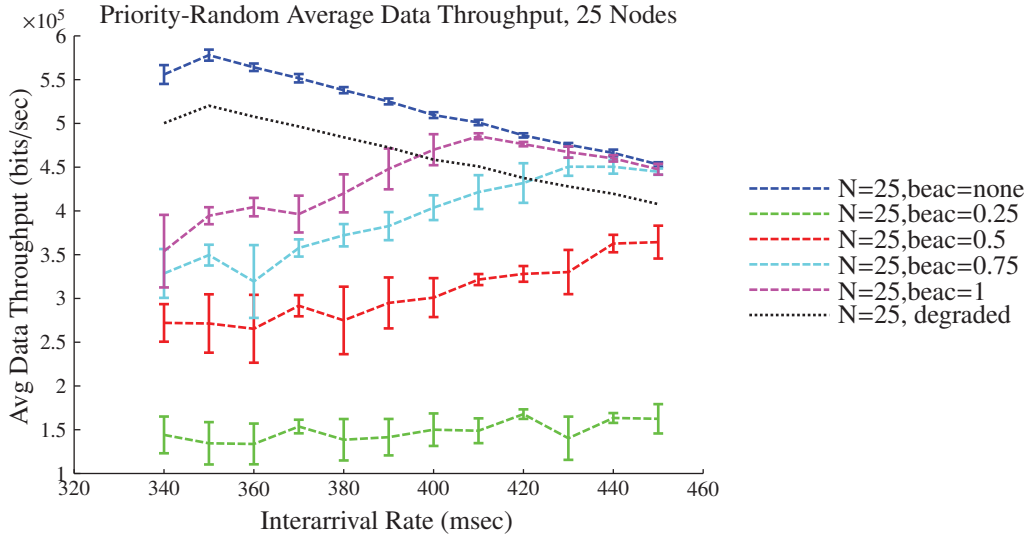


Figure A.244: N25 Data Throughput

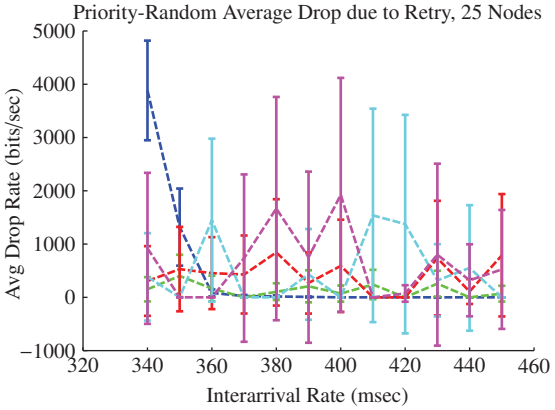


Figure A.245: N25 Retry Dropped Data

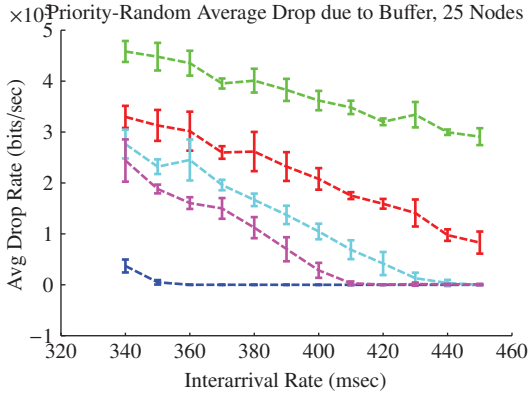


Figure A.246: N25 Buffer Dropped Data

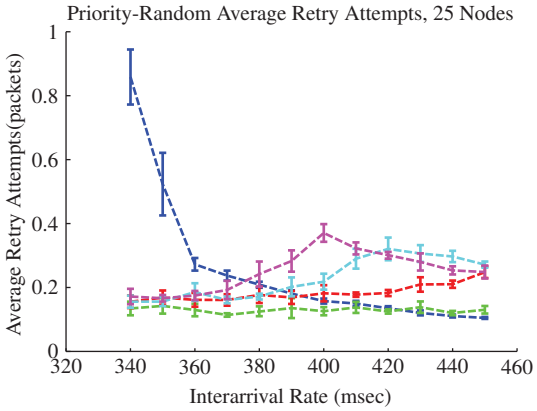


Figure A.247: N25 Retry Attempts

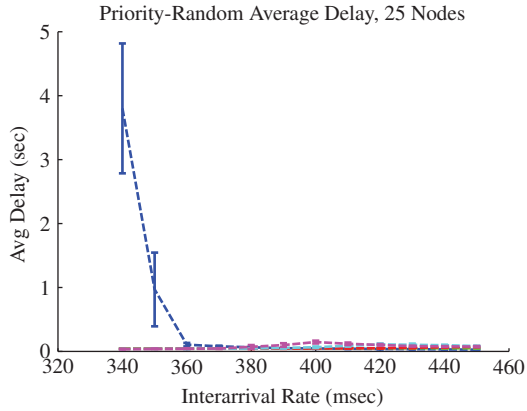


Figure A.248: N25 Packet Delay

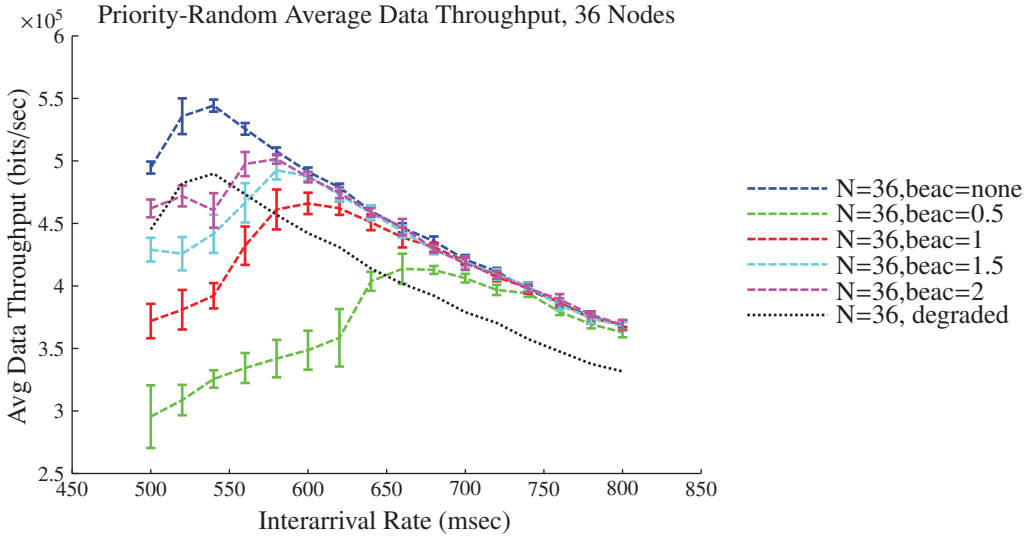


Figure A.249: N36 Data Throughput

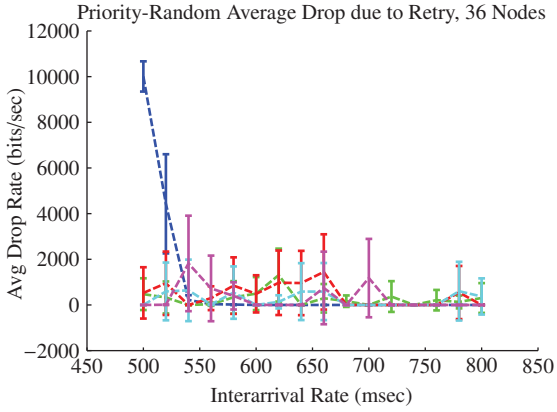


Figure A.250: N36 Retry Dropped Data

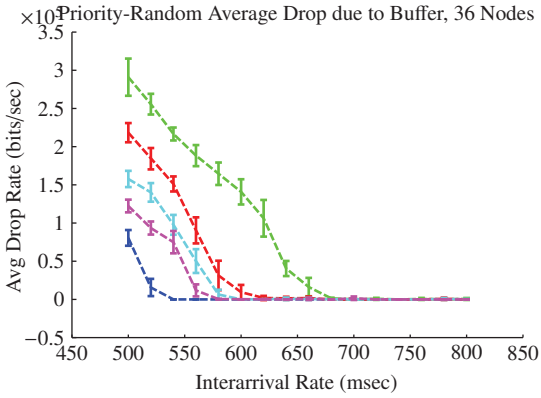


Figure A.251: N36 Buffer Dropped Data

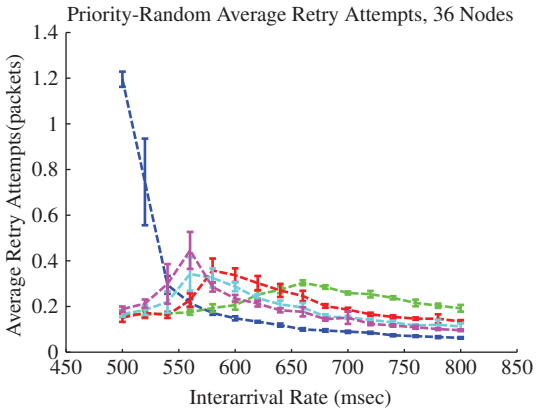


Figure A.252: N36 Retry Attempts

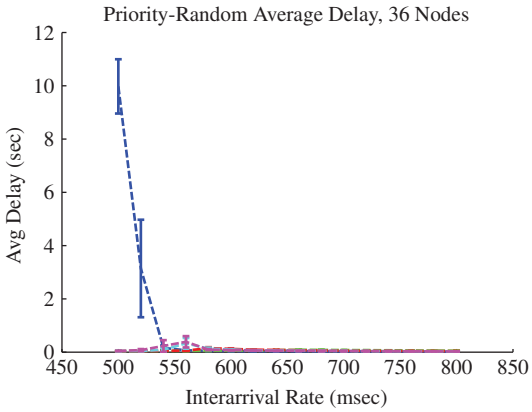


Figure A.253: N36 Packet Delay

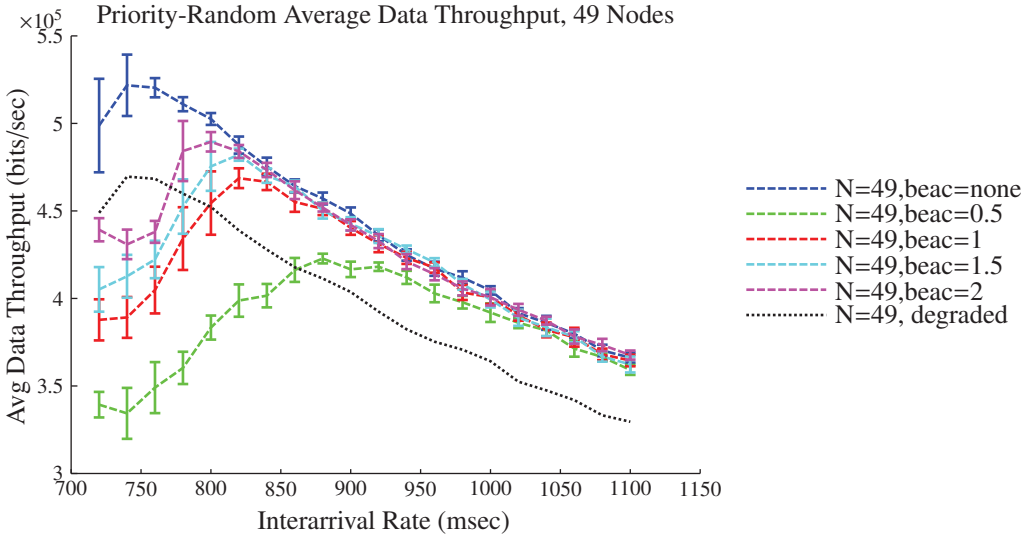


Figure A.254: N49 Data Throughput

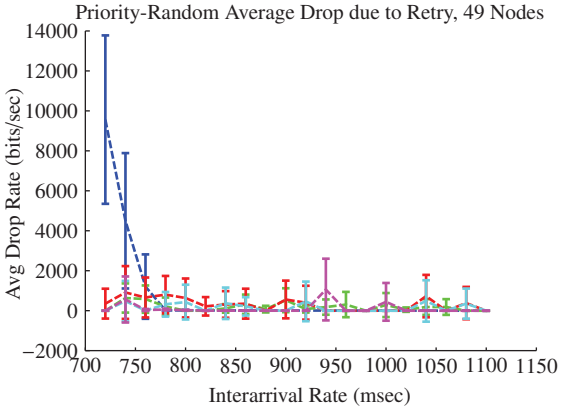


Figure A.255: N49 Retry Dropped Data

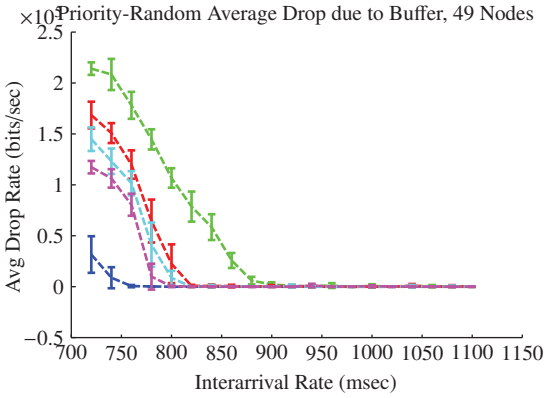


Figure A.256: N49 Buffer Dropped Data

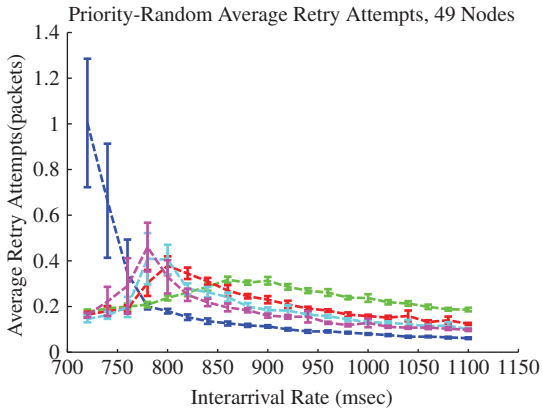


Figure A.257: N49 Retry Attempts

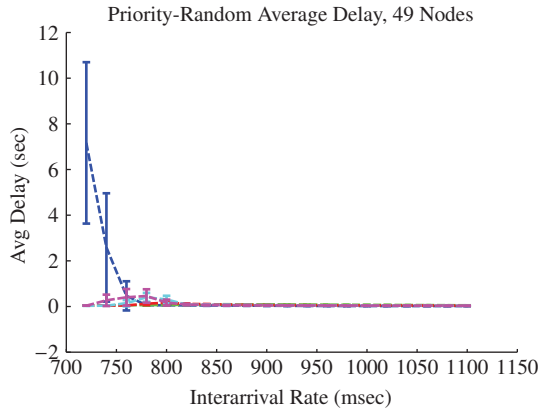


Figure A.258: N49 Packet Delay

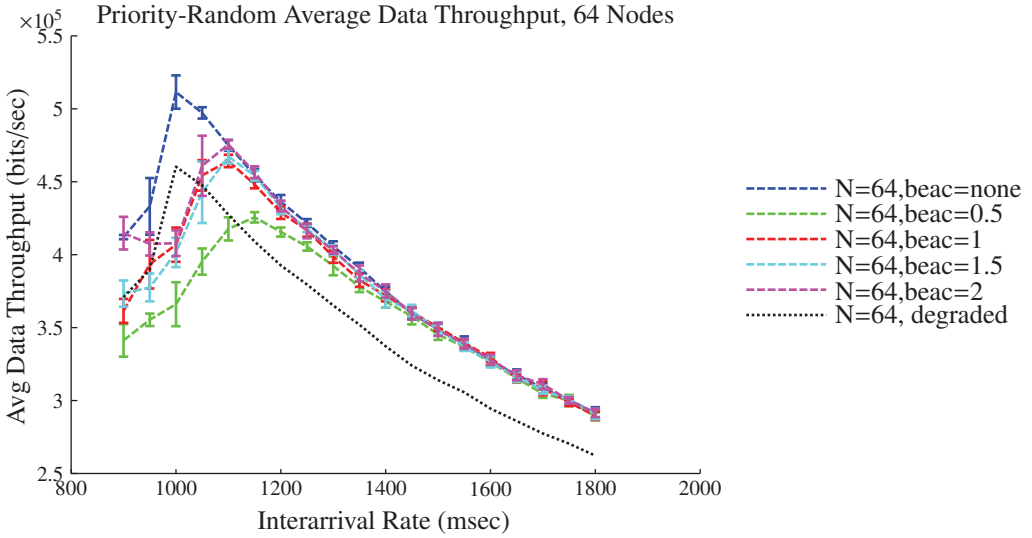


Figure A.259: N64 Data Throughput

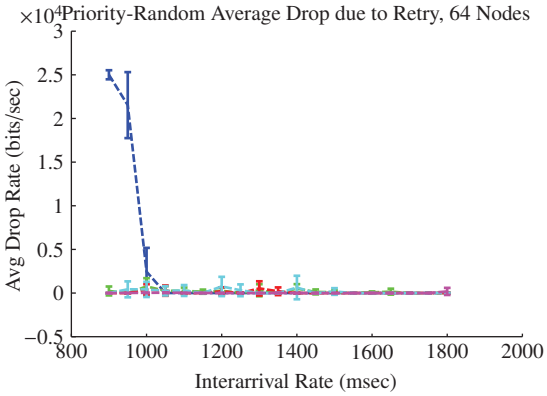


Figure A.260: N64 Retry Dropped Data

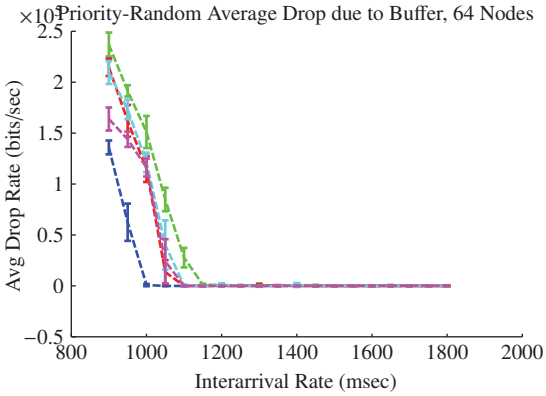


Figure A.261: N64 Buffer Dropped Data

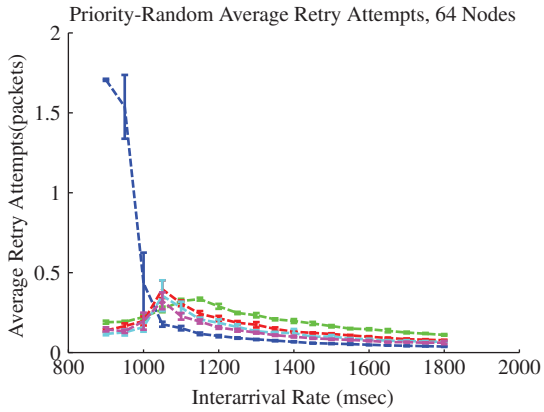


Figure A.262: N64 Retry Attempts

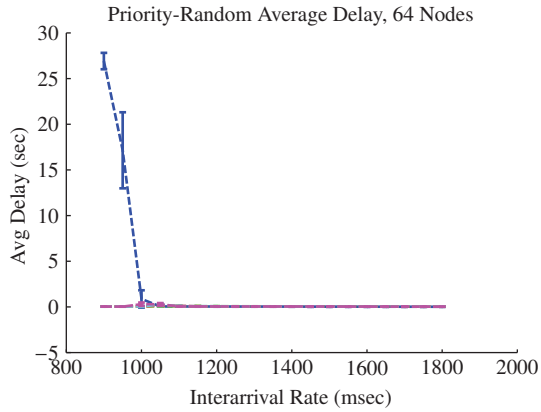


Figure A.263: N64 Packet Delay

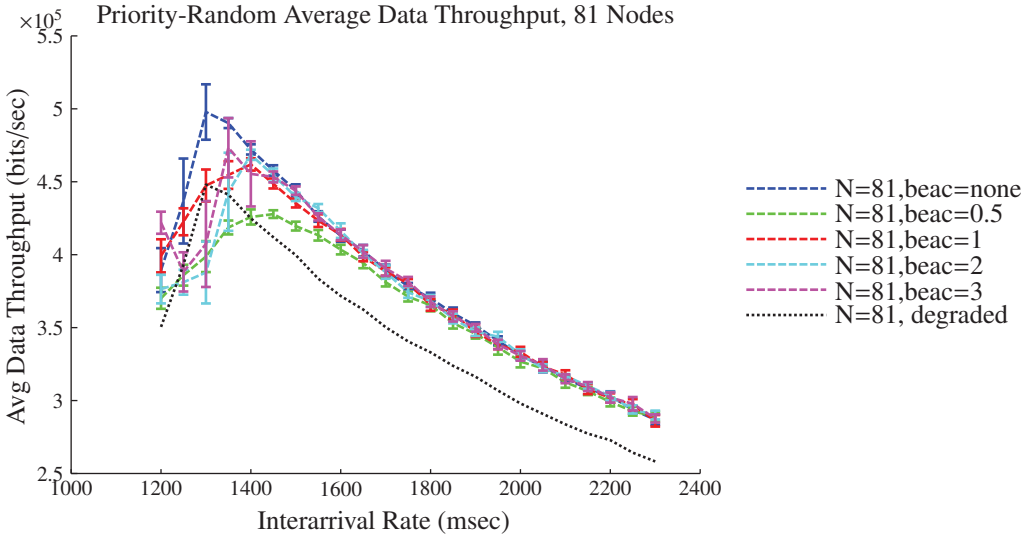


Figure A.264: N81 Data Throughput

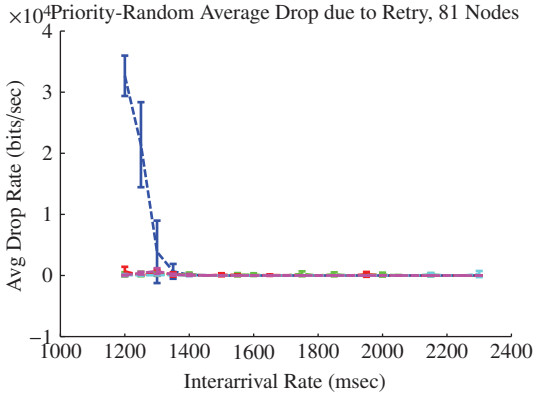


Figure A.265: N81 Retry Dropped Data

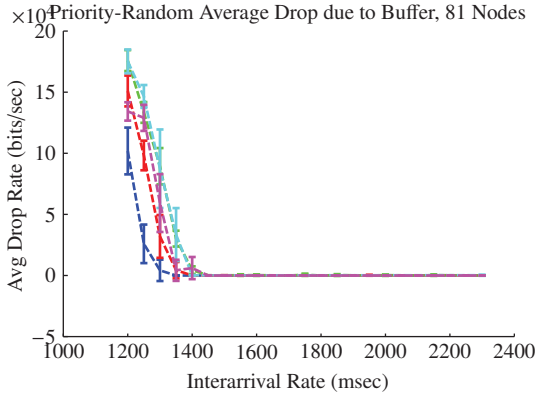


Figure A.266: N81 Buffer Dropped Data

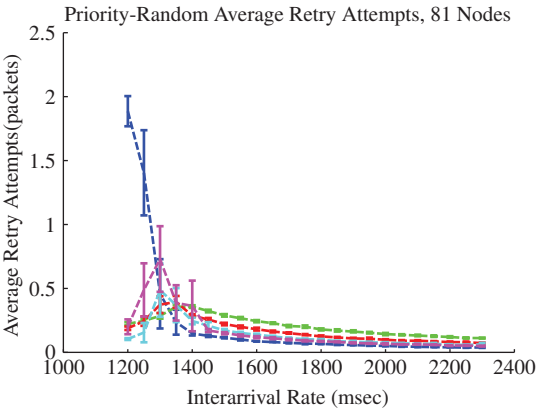


Figure A.267: N81 Retry Attempts

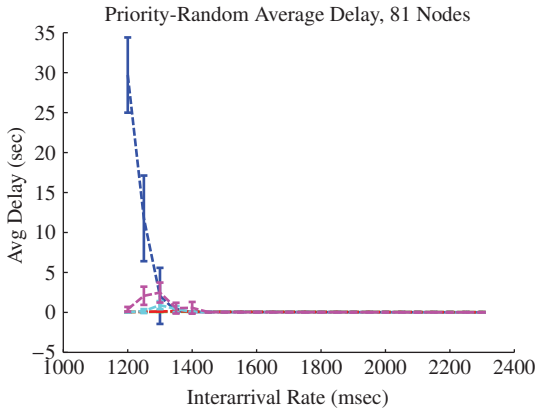


Figure A.268: N81 Packet Delay

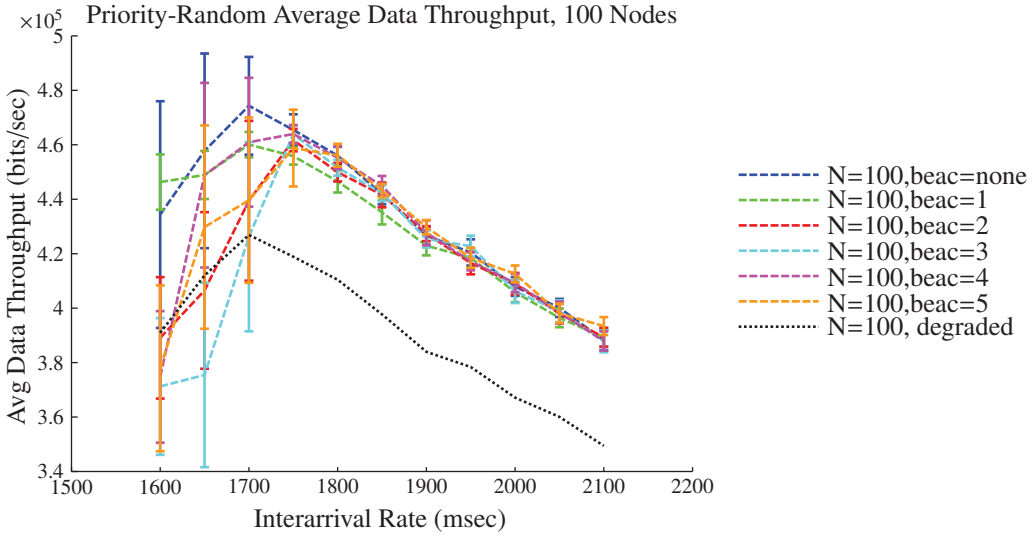


Figure A.269: N100 Data Throughput

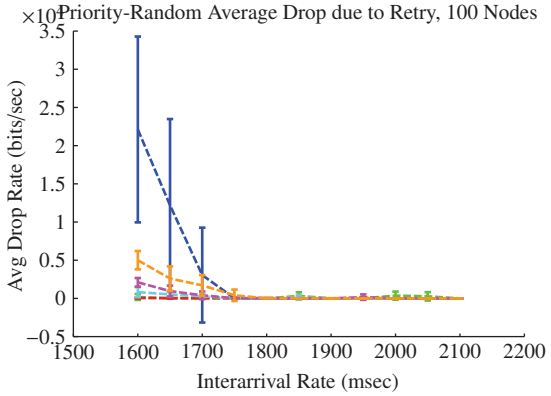


Figure A.270: N100 Retry Dropped Data

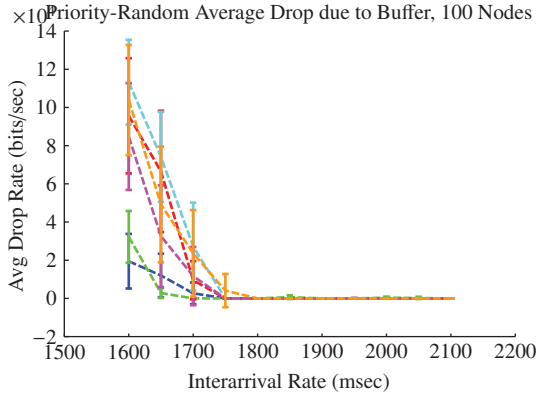


Figure A.271: N100 Buffer Dropped Data

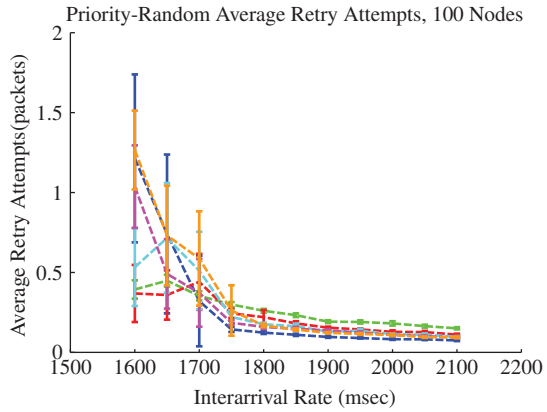


Figure A.272: N100 Retry Attempts

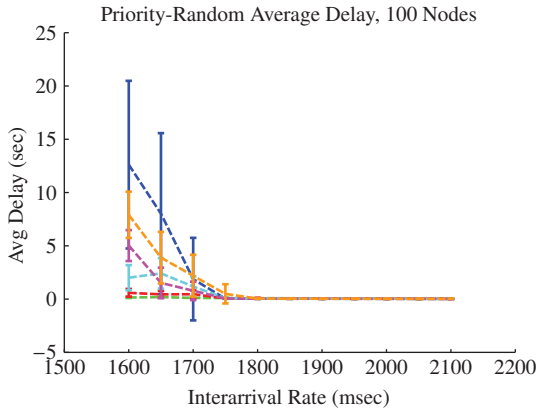


Figure A.273: N100 Packet Delay

A.5.4 Real World Scenario.

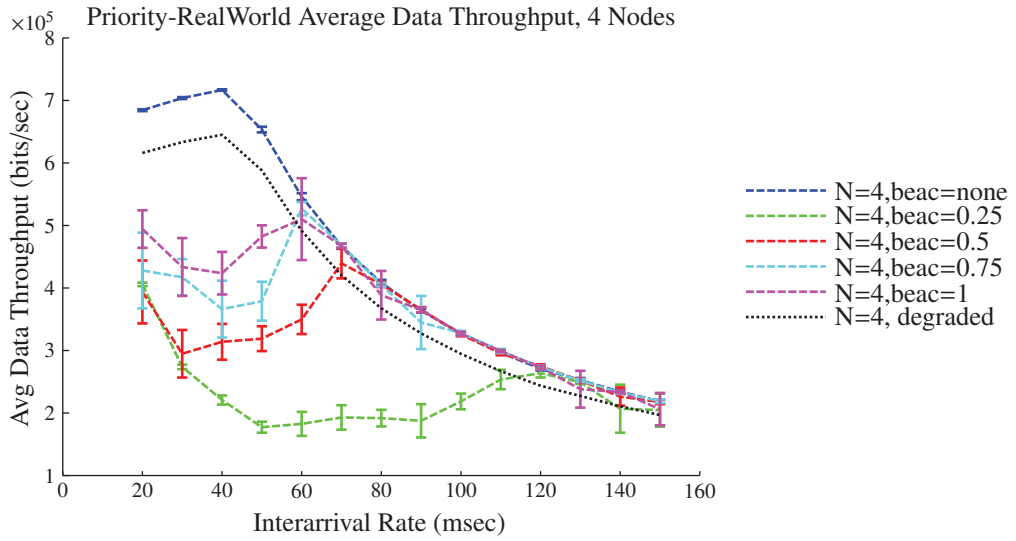


Figure A.274: N4 Data Throughput

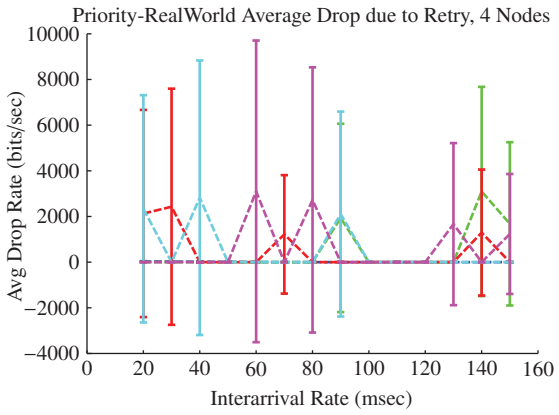


Figure A.275: N4 Retry Dropped Data

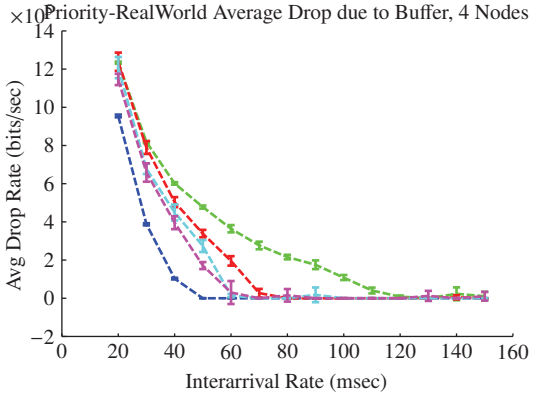


Figure A.276: N4 Buffer Dropped Data

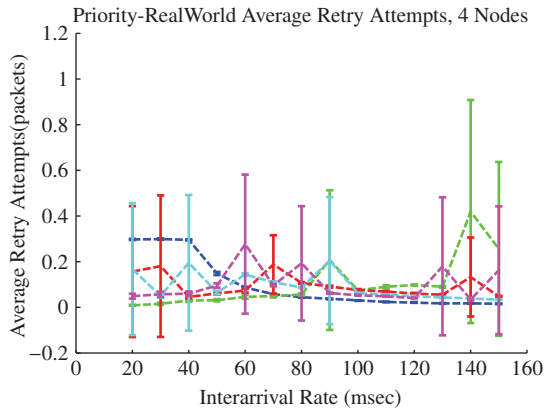


Figure A.277: N4 Retry Attempts

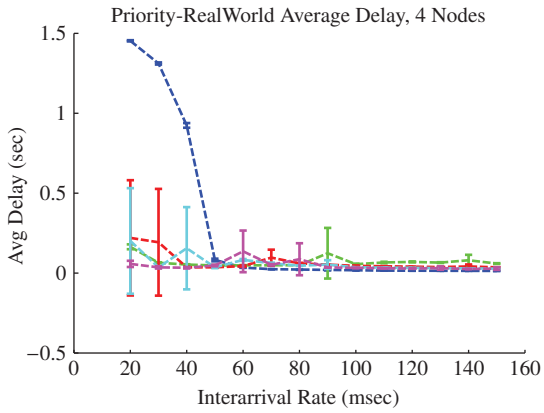


Figure A.278: N4 Packet Delay

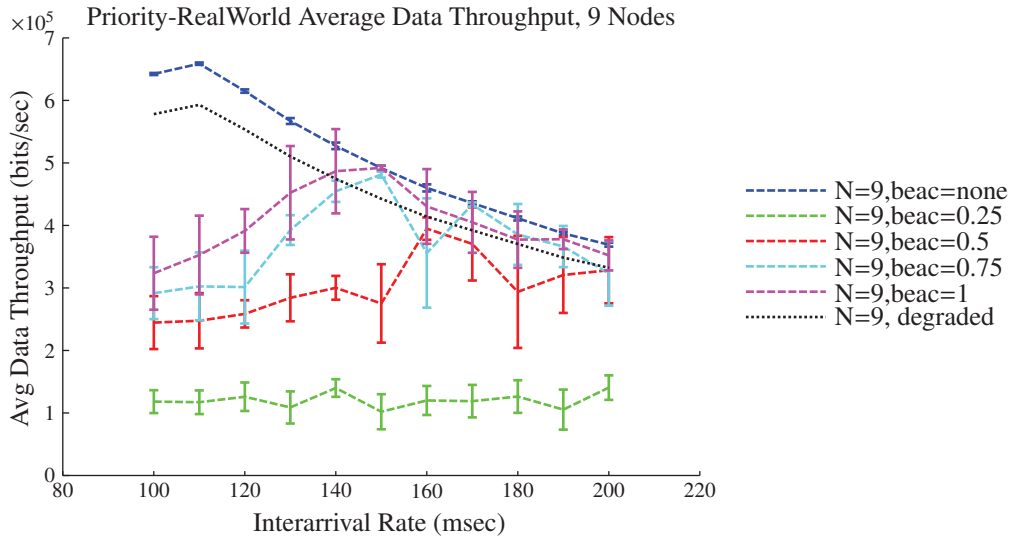


Figure A.279: N9 Data Throughput

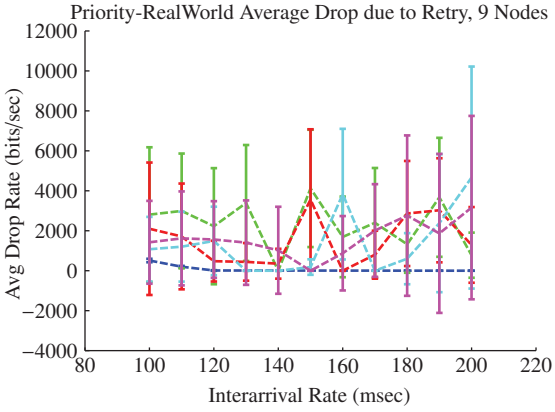


Figure A.280: N9 Retry Dropped Data

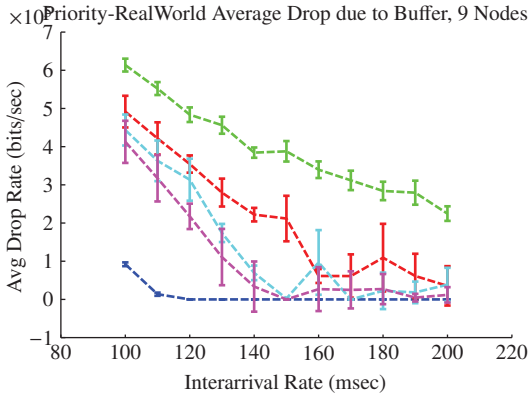


Figure A.281: N9 Buffer Dropped Data

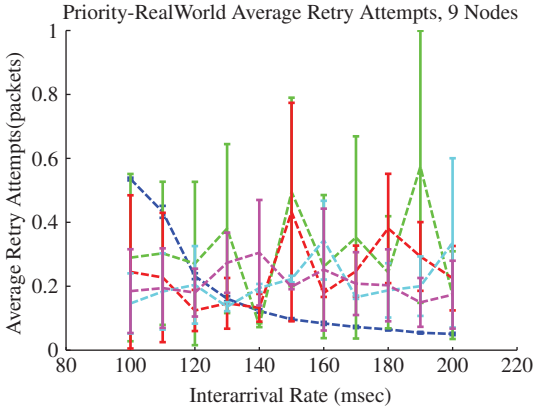


Figure A.282: N9 Retry Attempts

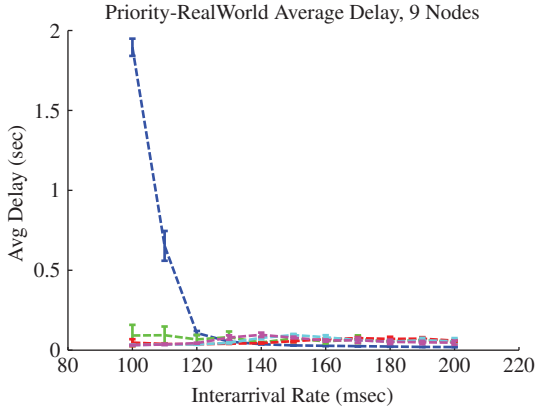


Figure A.283: N9 Packet Delay

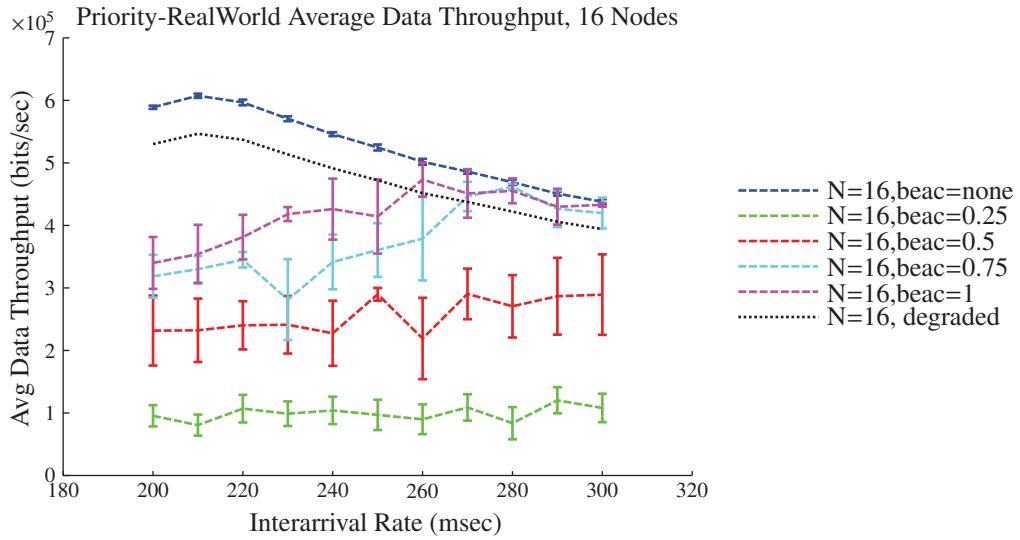


Figure A.284: N16 Data Throughput

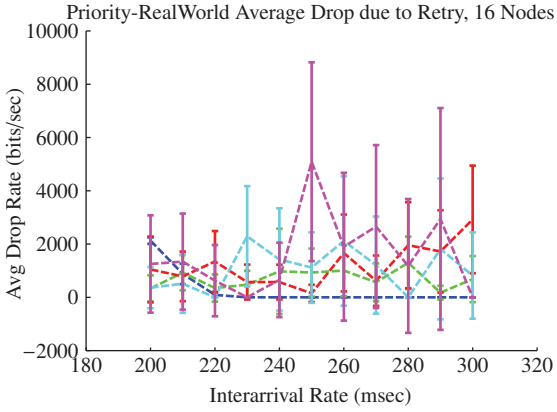


Figure A.285: N16 Retry Dropped Data

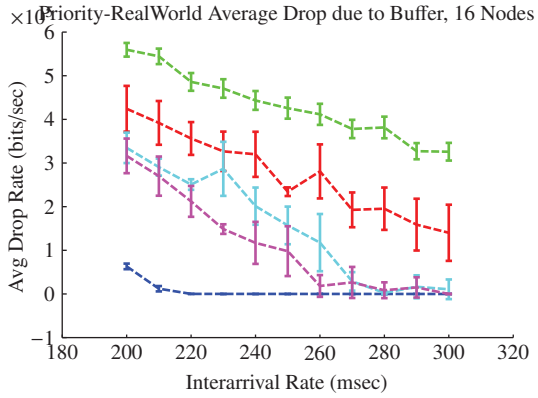


Figure A.286: N16 Buffer Dropped Data

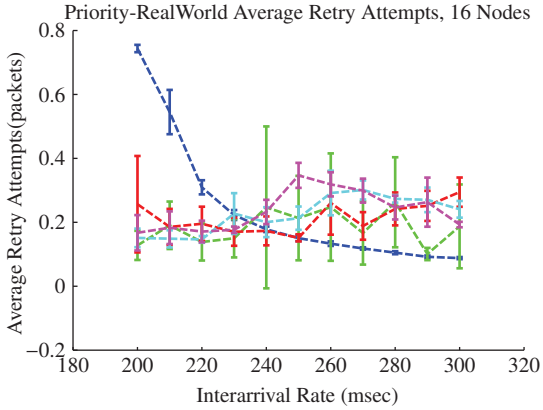


Figure A.287: N16 Retry Attempts

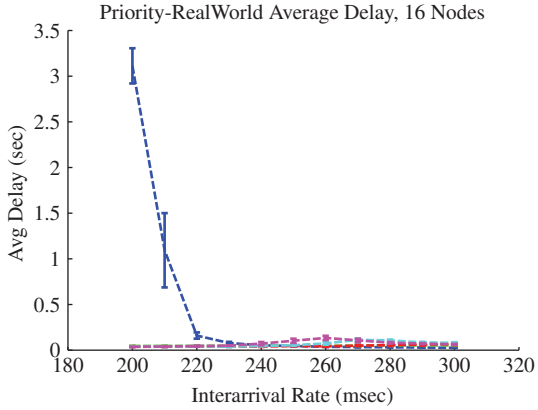


Figure A.288: N16 Packet Delay

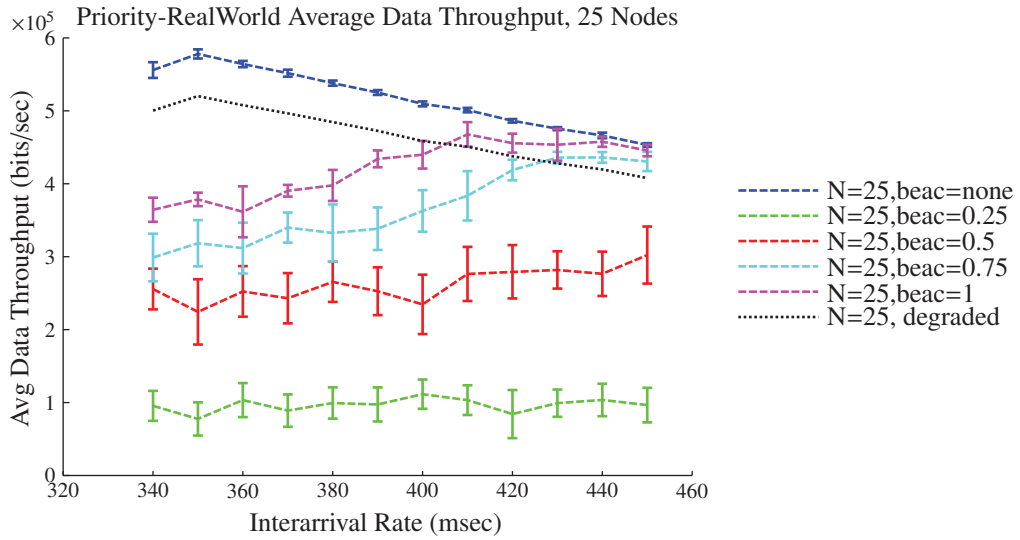


Figure A.289: N25 Data Throughput

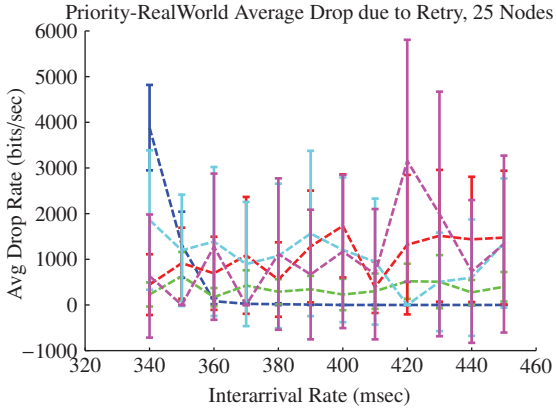


Figure A.290: N25 Retry Dropped Data

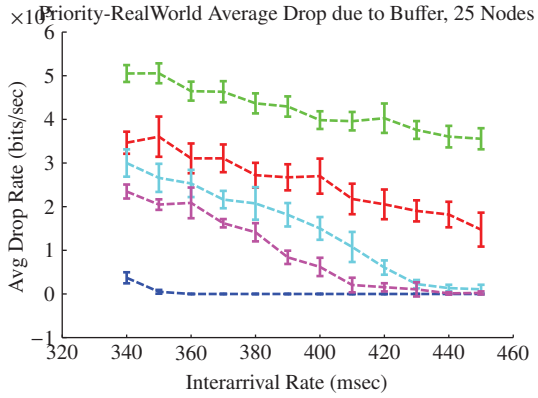


Figure A.291: N25 Buffer Dropped Data

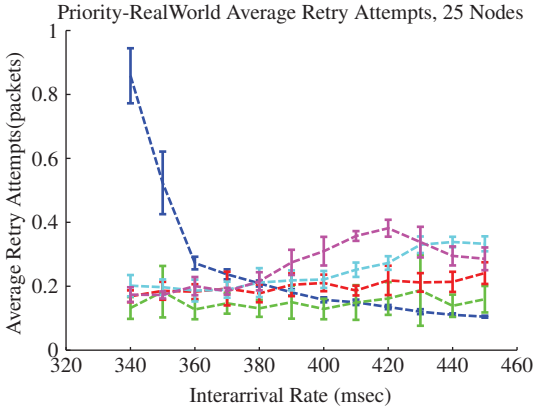


Figure A.292: N25 Retry Attempts

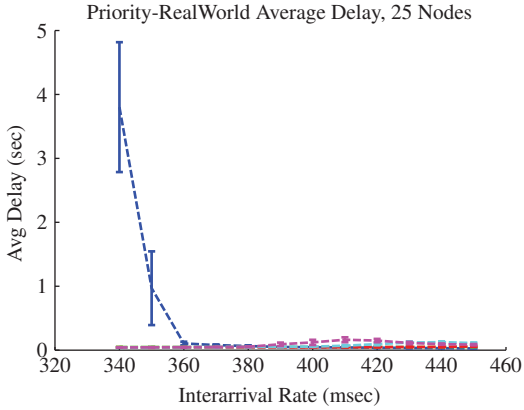


Figure A.293: N25 Packet Delay

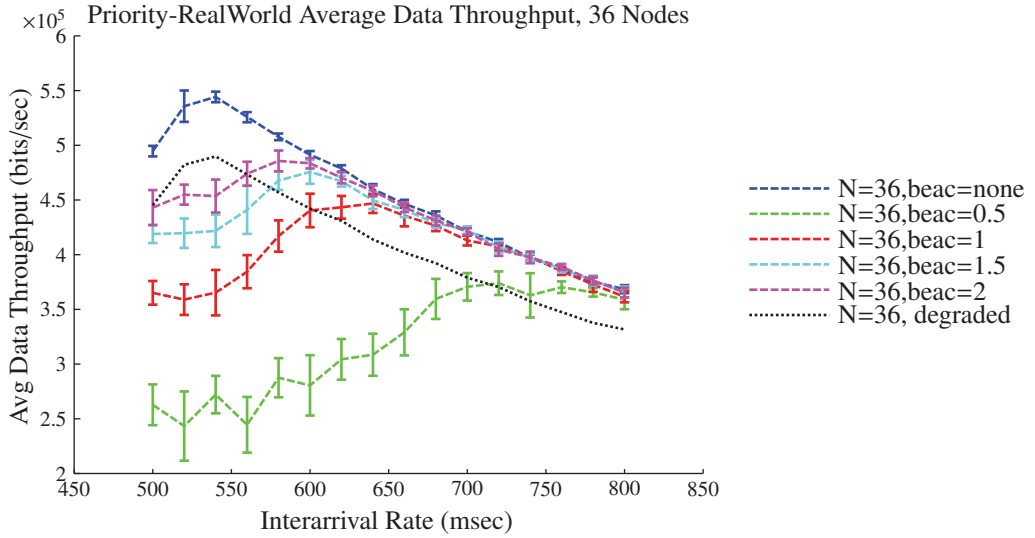


Figure A.294: N36 Data Throughput

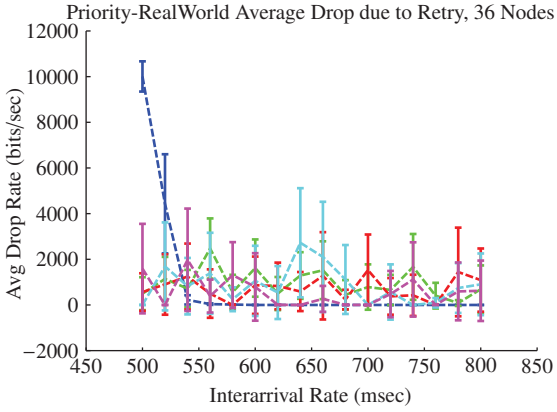


Figure A.295: N36 Retry Dropped Data

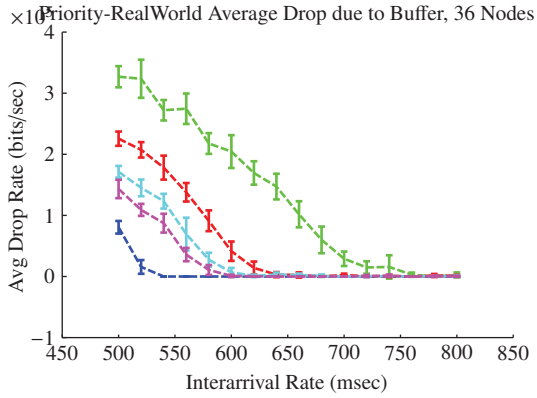


Figure A.296: N36 Buffer Dropped Data

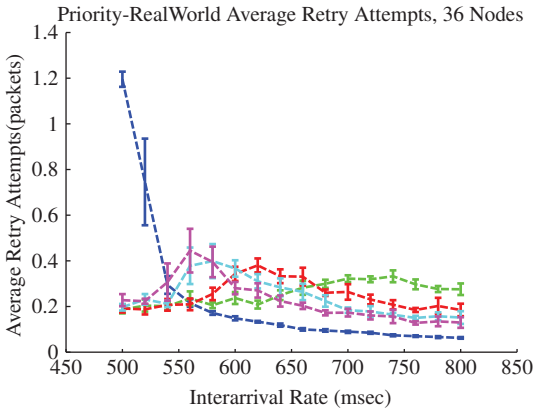


Figure A.297: N36 Retry Attempts

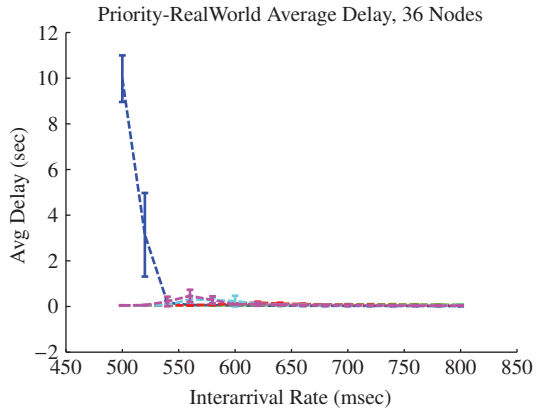


Figure A.298: N36 Packet Delay

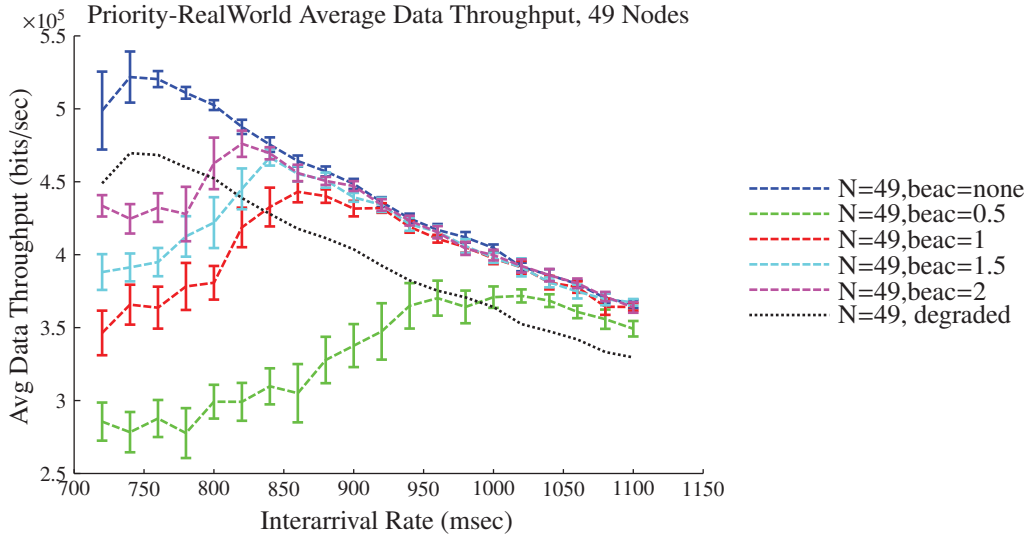


Figure A.299: N49 Data Throughput

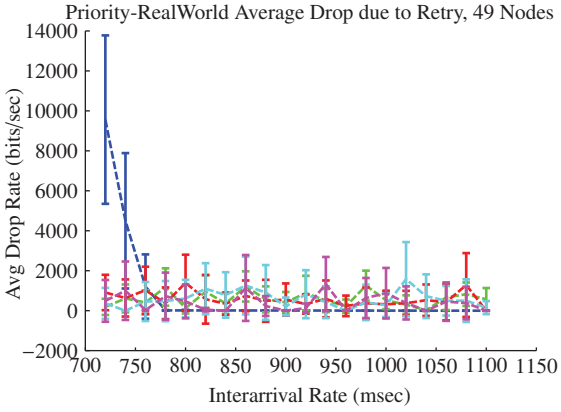


Figure A.300: N49 Retry Dropped Data

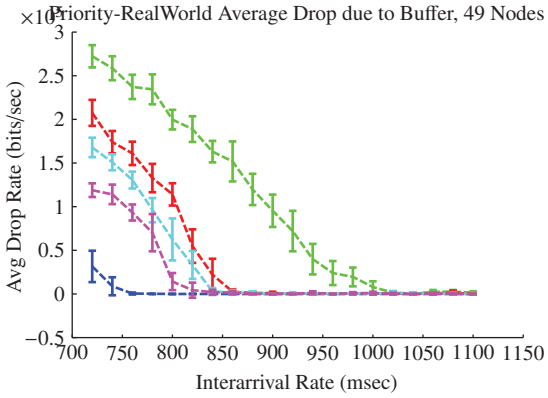


Figure A.301: N49 Buffer Dropped Data

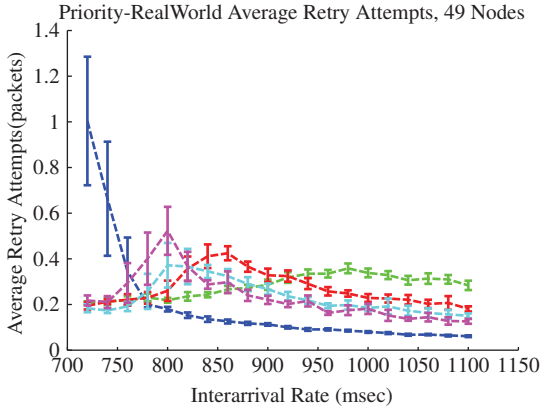


Figure A.302: N49 Retry Attempts

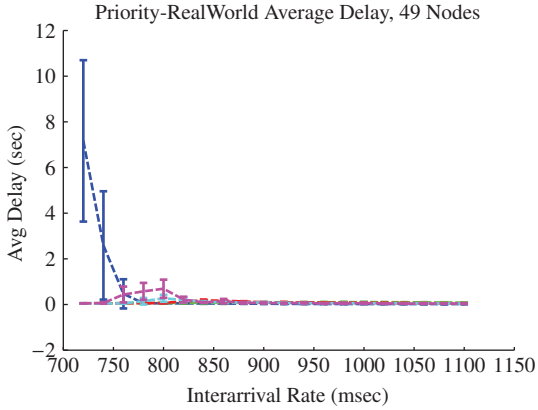


Figure A.303: N49 Packet Delay

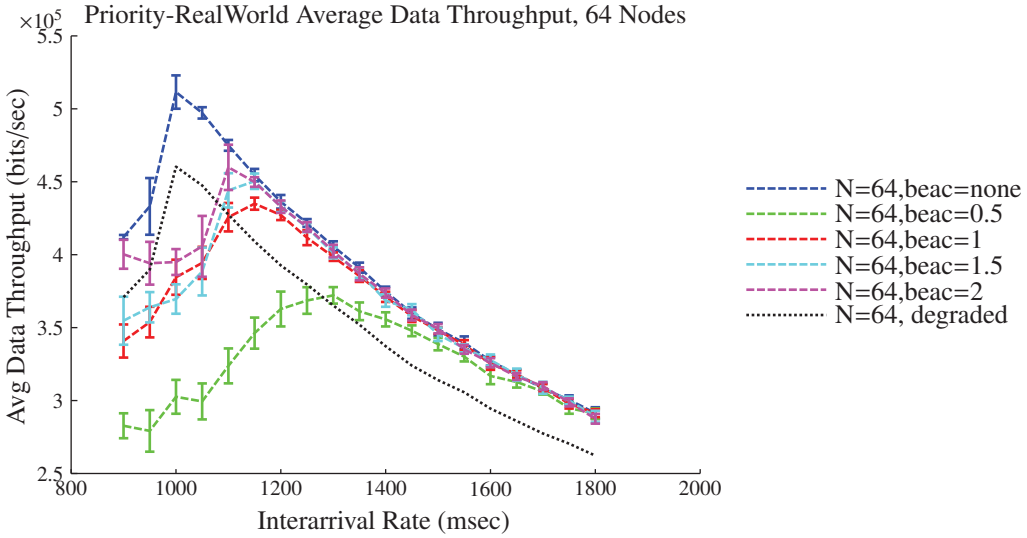


Figure A.304: N64 Data Throughput

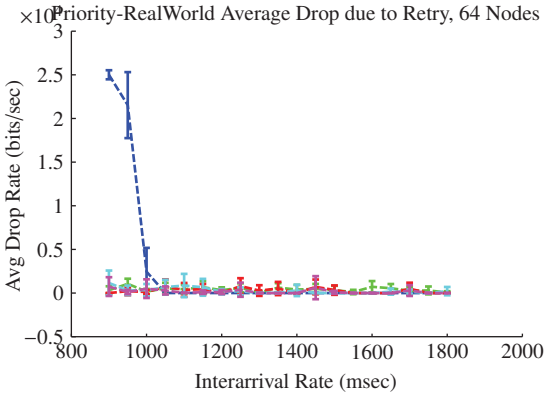


Figure A.305: N64 Retry Dropped Data

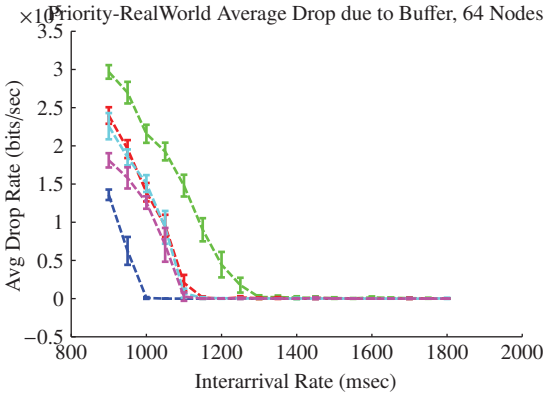


Figure A.306: N64 Buffer Dropped Data

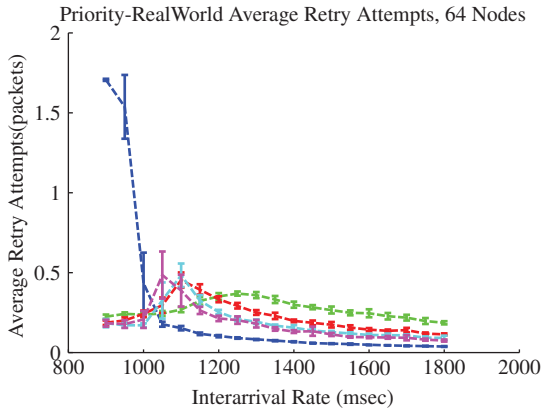


Figure A.307: N64 Retry Attempts

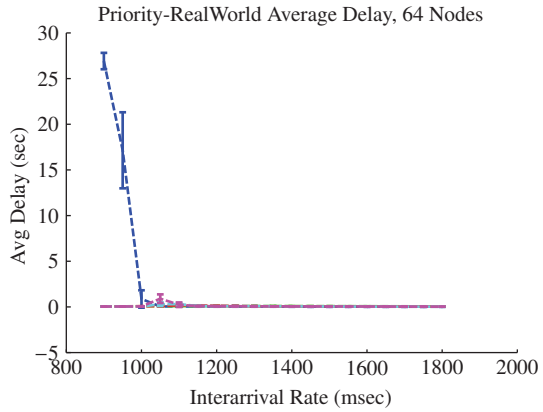


Figure A.308: N64 Packet Delay

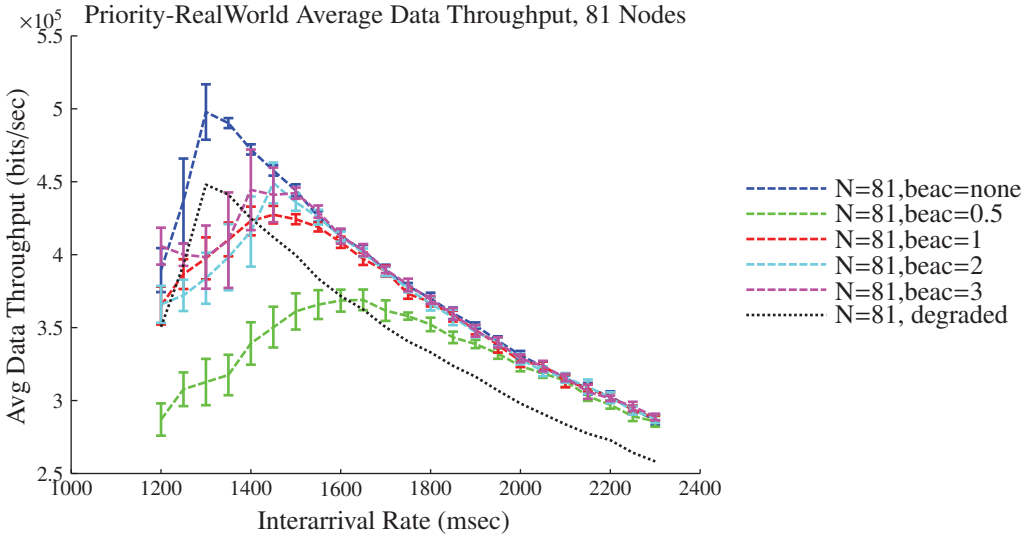


Figure A.309: N81 Data Throughput

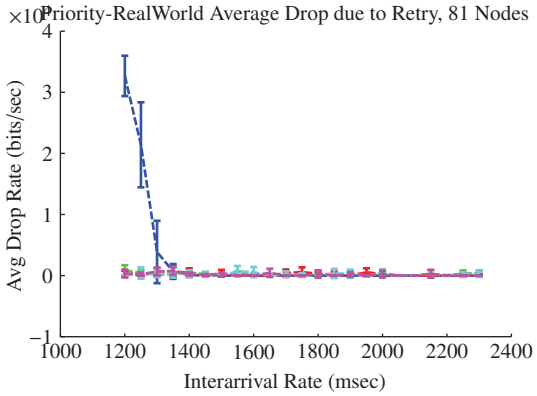


Figure A.310: N81 Retry Dropped Data

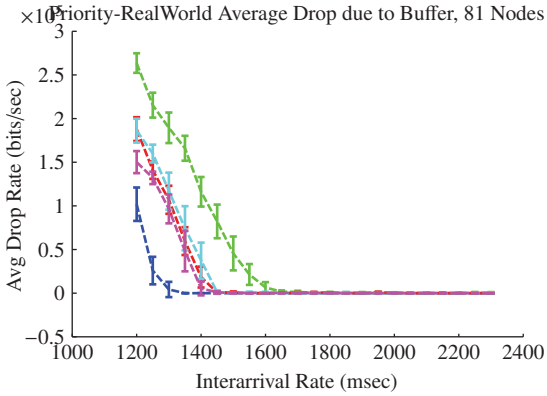


Figure A.311: N81 Buffer Dropped Data

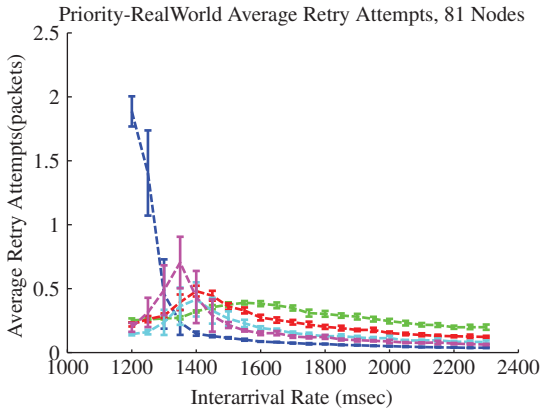


Figure A.312: N81 Retry Attempts

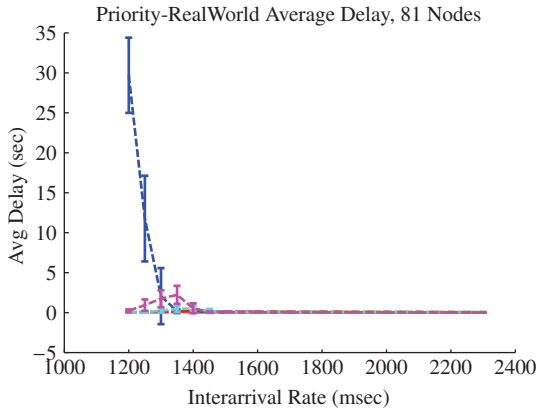


Figure A.313: N81 Packet Delay

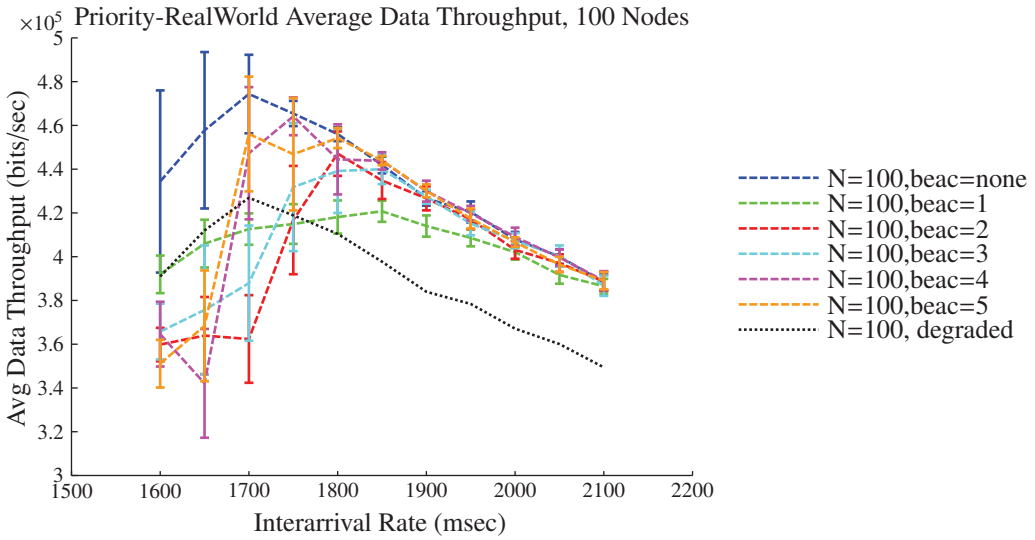


Figure A.314: N100 Data Throughput

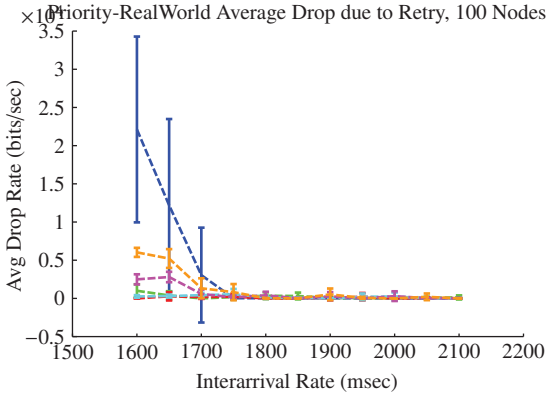


Figure A.315: N100 Retry Dropped Data

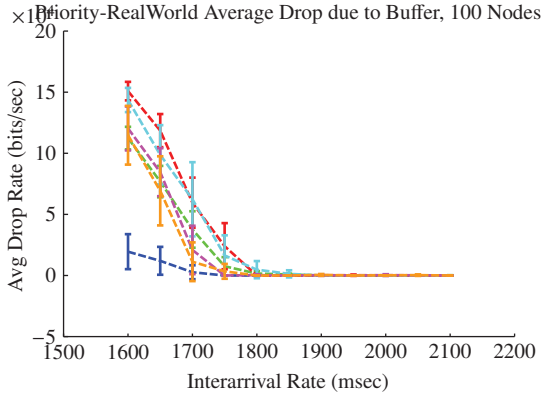


Figure A.316: N100 Buffer Dropped Data

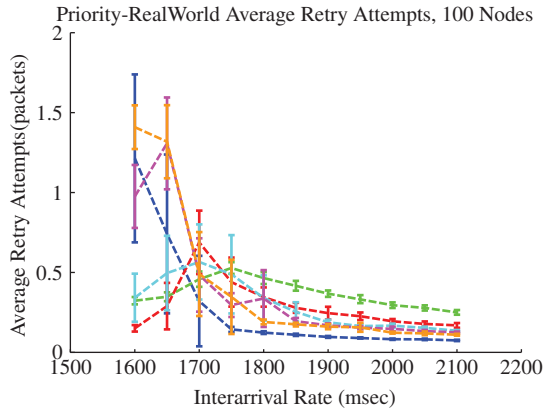


Figure A.317: N100 Retry Attempts

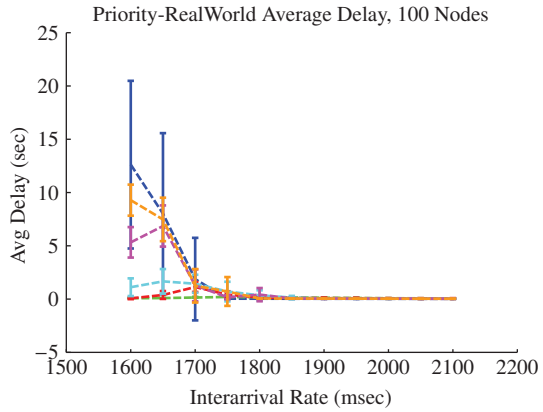


Figure A.318: N100 Packet Delay

Appendix: Gold's algorithm

B.1 Theory and Example

Gold's algorithm is based on the properties of a LFSR pseudorandom number generator (PRNG). The LFSR consists of a linear shift register of length N, frequency taps, and feedback taps. The number of frequencies taps is based on the number of frequency possibilities that the generator is to produce (i.e. for 32 frequencies, $log_2 32 = 5$, so 5 frequency taps are required). The feedback taps are used to create the feedback polynomial or reciprocal characteristic polynomial. The arrangement of taps for feedback in an LFSR can be expressed arithmetically as a polynomial modulo 2. This means that the coefficients of the polynomial must be 1s or 0s. This arrangement allows for the creation of the pseudorandom nature of the generator. The selection of tap points has been well studied and is out of the scope of this project. It should be noted that the selection of the feedback taps directly impacts the period of the sequence.

Gold's algorithm utilizes the linear relationship of the feedback and frequency taps. The algorithm executes three key functions to join an existing, transmitting system: 1) determination of a initial vector state for the receiver's LFSR; 2) determination of the register bits to use as frequency taps; and 3) recreation of and synchronization with the transmitting system's sequence.

To best understand the operation of Gold's algorithm, an example is used. The example was provided in the final report of the SIBR contract [28]. Understanding the operation of the algorithm is vital for VHDL implementation. The transmitter, or source, utilizes a 10-bit LFSR represented by Figure B.1. First, examine the changes in the LFSR over time. Let the LFSR be described by $\mathbf{a}$, a vector representing the feedback taps, and $\mathbf{z_i}$, the state of the LFSR at time slot i. The required a priori knowledge of $\mathbf{a}$ is a limitation to

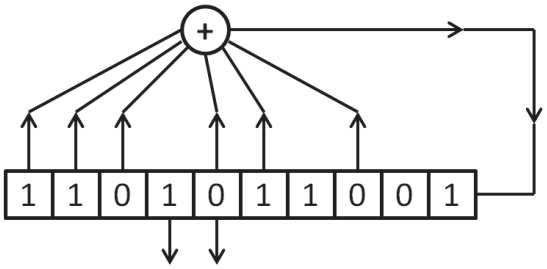


Figure B.1: Example Source LFSR PRNG

the algorithm.

$$\mathbf{a} = \begin{pmatrix} a_1 & a_2 & \dots & a_{10} \end{pmatrix}$$

$$\mathbf{z_i} = \begin{pmatrix} z_i & z_{i+1} & \dots & z_{i+9} \end{pmatrix}$$

For the example,

$$\mathbf{a} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

With this knowledge of **a**, a transition matrix can be created for the LFSR such that

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_2 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_3 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & a_4 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & a_5 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & a_6 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & a_7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & a_8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & a_9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & a_{10} \end{pmatrix} \tag{B.1}$$

This transition matrix is used to calculate the next state of the shift register, $\mathbf{z_{i+1}}$, given $\mathbf{z_i}$. This can be shown by

$$\begin{pmatrix} z_1 & z_2 & \dots & z_{10} \end{pmatrix} \times \mathbf{A} = \begin{pmatrix} z_2 & z_3 & \dots & z_{11} \end{pmatrix} \tag{B.2}$$

where $\mathbf{A}$ is the transition matrix. Additionally, it can be shown that any future register state can be calculated by using a known register state and the associative property of matrices.

$$\mathbf{z}_{i+2} = \mathbf{z}_{i+1} \times \mathbf{A} = \mathbf{z}_i \times \mathbf{A} \times \mathbf{A} = \mathbf{z}_i \times \mathbf{A}^2 \tag{B.3}$$

or $\mathbf{z}_t = \mathbf{z}_0 \times \mathbf{A}^t$ where $\mathbf{z}_t$ is the desired state, $\mathbf{z}_0$ is the original state, and t is the number of time slots between the occurrence of the two states.

Gold's algorithm functions by looking at the time of arrival of a single critical frequency by a receiver. By recording the time occurrences, a deterministic system can be created. For the example problem, the transmitter would create sequence 10110011011111111111000110001 at the two frequency taps. Let frequency 11 be the critical frequency of interest, and let the first instance of the critical frequency occur at $t = 0$. Therefore, in the above sequence the critical frequency occurs at $t = 0, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15$. Let these occurrences of the critical frequency be referenced as set T_n.

Then the matrices of interest are $A^{T_n} = \left\{A^0, A^4, A^7, A^8, A^9, A^{10}, A^{11}, A^{12}, A^{13}, A^{14}, A^{15}\right\}$. Note the algorithm requires N+1 occurrences of the critical frequency where N is the length of the LFSR.

The linear relationship between tap points on the source allows for another observation. Since the LFSR continues to shift the state left, a frequency tap of bit 4 and bit 5 is equivalent the taps being located at bits 1 and 2. In general, only the space relation between the frequency taps is critical, and as such the first tap can always be placed on bit 1 for the receiver. This aspect of recreating the LFSR solution allows us to show that all $\mathbf{z}_t$ share z_1 for the occurrences listed above.

$$\begin{pmatrix} z_1 & z_2 & \dots & z_{10} \end{pmatrix} \times A^{T_n}(1) = z_1 \tag{B.4}$$

where $A^{T_n}(1)$ is the first column of matrix A^{T_n} and the vector $\mathbf{z}$ is state of the LFSR at $t = 0$ (the first occurrence of the critical frequency).

Going back to the example, see that:

$$A^4(1) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad A^7(1) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad A^8(1) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad A^9(1) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad A^{10}(1) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$A^{11}(1) = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad A^{12}(1) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad A^{13}(1) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad A^{14}(1) = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad A^{15}(1) = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{B.5}$$

By collecting these columns together, the following equation emerges.

$$\begin{pmatrix} z_1 & z_2 & \dots & z_{10} \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} = 0 \tag{B.6}$$

A transposition of the equation yields a system of equations to create the solution space.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \\ z_8 \\ z_9 \\ z_{10} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{B.7}$$

To solve the system of equations, binary row reduction is used to obtain the canonical form.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_6 \\ z_7 \\ z_8 \\ z_9 \\ z_{10} \\ z_4 \\ z_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{B.8}$$

which equates to $[z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, z_{10}] =$

$[z_5, z_4 + z_5, 0, z_4, z_5, z_4, z_5, z_4, z_5, z_5, z_5]$

Note that the solution space depends on two variables, z_4 and z_5. The n independent variables are directly related to the use of n frequency taps where $n = 2$ in this example. Manipulating these variables creates the vector space of candidate solutions shown in Table B.1.

z_4	z_5	z_1,	z_2,	z_3,	z_4,	z_5,	z_6,	z_7,	z_8,	z_9,	z_{10}
0	0	0,	0,	0,	0,	0,	0,	0,	0,	0,	0
0	1	1,	1,	0,	0,	1,	1,	0,	1,	1,	1
1	0	0,	1,	0,	1,	0,	1,	1,	0,	0,	0
1	1	1,	0,	0,	1,	1,	0,	1,	1,	1,	1

Table B.1: Vector Space of Candidate Solutions

An all-zero solution is not a satisfactory solution since it always provides a feedback of zero; therefore, only three possible solutions exist. Next, the algorithm identifies any shift relationships among the solution candidates to determine which of the solutions is the unique solution.

Examining shift relationships shows that solution $(1, 0, 0, 1, 1, 0, 1, 1, 1, 1)$ is same as the single left shift of $(1, 1, 0, 0, 1, 1, 0, 1, 1, 1)$. To find the unique solution, this method of shift comparison requires that n solutions (the same as the necessary number of taps) be found within a single candidate vector. In this case, $(1, 1, 0, 0, 1, 1, 0, 1, 1, 1)$ has proven that itself and one other solution are contained within it, and therefore must be the unique solution.

Additionally, the number of shifts required to create the other candidate solutions indicates the frequency tap points. Specifically, the first bit in the unique solution is a frequency tap since the unique solution matches itself. The second solution is created by one shift; therefore, the second bit is also a frequency tap.

Through these calculations, two of the three functions of the algorithm have been achieved: 1) determination of an initial vector state for the LFSR; 2) determination of the register bits to use as frequency taps. The third function, recreation of and synchronization with the transmitting system's sequence, only requires loading the LFSR with the calculated initial value and stepping through the same number of frequency changes that the source progressed through since the first occurrence was recorded.

B.2 VHDL Implementation

The hardware implementation consists of a 16-bit length LSFR with 5 feedback taps and 5 frequency taps. As such, the PRNG will generate a sequence spanning 32 frequencies according to a near uniform distribution. The register length and selection feedback tap locations determine the period of a repeating pattern with in the sequence. The maximum period for a 16-bit LSFR is 65,535 cycles. Gold's algorithm, as written in

VHDL, consists of seven distinct computation processes, or blocks, and a top-level wrapper used for component coordination and control. This code comprises the custom IP core used in this research effort. The design requires as inputs: one clock operating at the same rate which the transmitter is changing frequencies; one higher rate clock constrained by timing requirements of the hardware implementation; a critical frequency detection core; and software registers to control initial values and an asynchronous reset. Figure B.2 shows a graphical representation of the Gold's algorithm custom IP core broken into processing blocks.

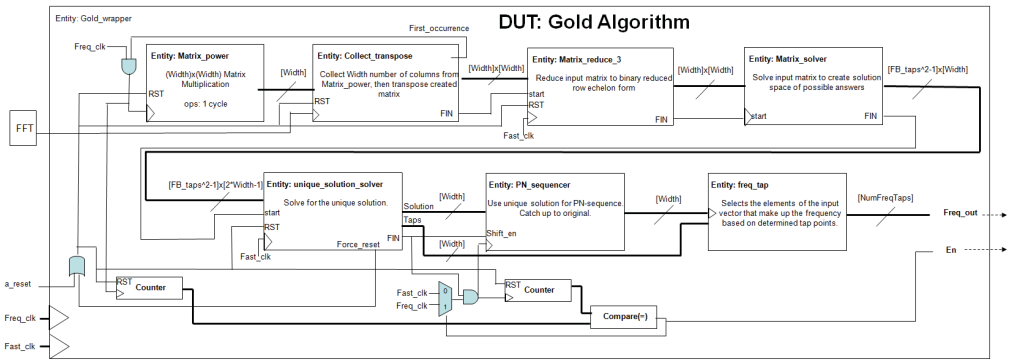


Figure B.2: Gold's algorithm custom IP core

B.2.1 Top-level.

The top-level wrapper coordinates block communication and provides control logic for operations. First, the system determines when the critical frequency occurs. A combination of a Xilinx FFT IP core and a simple signal comparison accomplishes this task. When the frequency designated as the critical frequency is detected, the comparison produces a trigger which is used by Block 2, as described in Section B.2.3 Two resets exist for the system, a software register commanded reset and a hardware reset from Block 5.

The controls incorporated into the top-level wrapper primarily operate to allow the system to correctly synchronize after a solution has been found. Since the system works by calculating the LFSR at the first occurrence of the critical frequency, the number of

frequency changes beyond this $t = 0$ must be tracked. A counter (referred to as the primary counter) tracks these changes. Once the system determines a unique solution, it must progress through the same number of changes as the transmitter. The use of a second counter tracks the LFSR changes in the receiver. Lastly, a comparator determines when the two counters match. This comparator serves as a synchronization indicator. Additionally, this synchronization indicator determines the clock rate used in Block 6, as described in Section B.2.6

B.2.2 Block 1: Matrix power multiplication.

Block 1 of the VHDL code implements the transformation matrix multiplication once for every rising edge of the frequency change clock. The transition matrix is created using an input vector for $\mathbf{a}$, as in Equation B.1. This vector loads from a software register allowing for quick adaptation. An intermediate matrix is stored each clock cycle, and is subject to a system reset input. Block 1 outputs a vector containing the first column of the calculated matrix as shown in Equation B.5. This vector is used by Block 2. Block 1 executes in 1 cycle.

B.2.3 Block 2: Collection and transposition.

Block 2 collects the column vectors from Block 1 when a critical frequency is detected. The frequency detection signal is externally triggered by an FFT. As this block latches the column vector from Block 1, it also performs the transpose operation converting each column into a row, and combining the rows into a square matrix which is the output of the block. This operation produces results similar to Equation B.7. Once Block 2 detects and collects $N + 1$ vectors, it triggers a finish flag to initiate Block 3. Block 2 executes in 1 cycle.

B.2.4 Block 3: Matrix reduction.

Block 3 preforms a binary row echelon matrix reduction on the matrix from Block 2. Since the matrix reduction process depends on the only the completion of Block 2, a faster

clock rate can be utilized. The faster clock rate is used on all further blocks in the system. The reduction process uses a 7 state finite state machine (FSM) design. The output is a row echelon reduced matrix similar to Equation B.8

State 0 is used as an initial load state, and is only used upon block initialization.

State 1 is used to detect the first occurrence of a *1* in the first column of the matrix. The first entry (upper right of matrix) is examined to determine if it is a 1. If it is a *1* and another *1* had not already been found, the entire row is stored as a temporary signal vector. The stage will then rotate the whole matrix by one row (i.e. the first row is shifted to the bottom and row 2 is moved to row 1). This rotation occurs for each row whether nor not the row was selected as the temporary signal vector. Therefore, State 1 requires N cycles to complete. After the completion of State 1, the hardware transitions to State 2.

State 2 is similar to State 1 in that it rotates through each row detecting when a *1* occurs in at the first entry of the matrix. The difference occur in the operation after detection. If a row is detected to have a left most *1* then the row is compared to the temporary signal vector. This test eliminates the ability to manipulate the row with the first *1* occurrence. Other rows that contain a *1* will undergo a bitwise xor operation between that row and the temporary signal vector. This serves to eliminate all but the first *1* in the first column of the matrix. State 2 requires N cycles to complete. After the completion of State 2, the hardware transitions to State 3.

State 3 performs a row swap operation on the matrix. The operation swaps the row that had the first *1* occurrence from Stage 1 with the top row. This stage requires 1 cycle to complete.

State 4 rotates the matrix one more time by moving the top row to the bottom of the matrix, and all other rows up by one. This stage requires 1 cycle to complete.

State 5 performs a matrix left rotate operation. Each row is individually rotated such that its first bit becomes its last bit while all other bits shift left by one position. Stage 5

will run N times so that each column is examined by states 1 through 4. Therefore, if the stage has executed less than N times the state is reset to state 1. Otherwise, the state is progressed to State 6. Each execution of Stage 5 requires 1 cycle, but also controls the re-execution of previous states.

State 6, the last state in the block latches the result from the intermediate matrix to the output matrix to be used by the next block. It also sets a finish flag to initiate Block 4 to begin operations.

The total execution requirement of block 3 is $2N^2 + 3N + 2$ cycles, where N is the size of the LFSR.

B.2.5 Blocks 4 and 5: Solution solver.

To decrease area usage and take advantage of internal block RAM, blocks 4 & 5 are combined to form a single component.

Block 4 preforms the matrix solver process. This block takes the reduced matrix, determines which bits are the independent variable, and creates a solution space based on the possible solutions. The solving process uses a four state FSM design.

State 0 is used as an initial load state, and is only used upon block initialization.

State 1 determines which bits are the independent variables. In the matrix, any row that contains all *0*s is defined as an independent variable. As such, state 1 performs a comparison on each row. The number of variables is equivalent to the number of frequency taps. The bit location of each is tracked using a vector signal. State 1 requires $N + 1$ cycles to complete. The extra cycle presets signals to be used in state 2.

States 2 and 3 work in combination to populate the solution space matrix based on the permutations of the independent variables. For each possible combination (i.e. 0001, 0010, 0011, etc.) of the variable bits as determined by stage 1, the reduced matrix is evaluated to produce a candidate solution for the solution space. The evaluation utilizes a distinct combination of *and* and *xor* operations in sequence for each row. Each candidate

solution is loaded into the block RAM upon calculation. The all-zero combination is removed giving $2^{freqtaps} - 1$ possible solutions in the solution space. State 2 requires 1 and State 3 ($N + 1$) cycles to complete for each possible combination of the variable bits. At the end of processing Block 4, the candidate solutions have been determined. This results in the block RAM containing information similar to Table B.1.

Block 5 evaluates the solution space to determines if a unique solution exists. If a unique solution cannot be identified, the block commands a system reset to the Gold's algorithm IP core. The unique solution solving process uses a six state FSM design.

State 4, a continuation from State 3 in Block 4, load temporary signals with initial values, and is only executed upon block initialization. For initialization, the state loads the first entry from block RAM to a temporary vector which will be used for comparisons. This temporary vector is updated throughout allowing different values to be compared. State 7 controls the value of vector as needed.

State 5 compares the bits of the temporary vector with each of the other candidate solutions. The result of each row's comparison are tracked by a separate vector to be analyzed in State 6.

State 6 uses the comparison result vector from State 5 to perform two actions upon a detected match. First, each candidate solution when evaluated uses a counter to determine if the number of other solutions are equal to the number of frequency taps. As such the the comparison vector indicates when to adds to the counter. Secondly, a position tracker records which of the other solutions was the match. After the first occurrence of State 6, the first candidate solution without any bit shifts has been loaded as the temporary vector and compared to the other solutions. The counter results should indicate that the temporary vector matched only itself. However, Gold's algorithm requires each candidate solution must compare left shifted versions of itself to the other possible solutions too. The shifting process is controlled in State 7.

State 7 determines when the unique solution has been found, and controls the loading of the temporary vector for further comparisons. The control state first determines if any of the tested solutions already meet the requirement to be deemed unique (i.e. contains x number of solutions within it, where x is the number of frequency taps needed). If a solution has not been identified, the temporary vector is updated. Two conditions exist for updating the vector. First, as stated, each solution must compare several left shifted versions of itself to the other possible solutions. Each solution can be used to create N possible shifted versions. A shift is created by using the known feedback information the currently value to determine the need an extension bit. After each shifted version has been tested and it is determined that the candidate solution is not the unique solution, then the temporary vector is updated with the next candidate solution from the block RAM. After each update to the temporary vector the state of the FSM is set back to State 5. If the controller determines a unique solution exists at any time then no other solutions are tested. The discovered unique solution along with the position vector for that solution are latched to outputs vectors. The position vector represents the bit locations that must be frequency taps. The unique solution output is directed to block 6 while the position vector is routed to Block 7.

State 8 determines if the unique solution is valid. Error in validity can occur if the system obtained a false positive or an error in calculations occurred. If the solution is not valid, the reset signal is set; otherwise, a finish signal initiates Block 6.

In the worst case, a solution is not detected and this block must compare each possible solution to each possible shift location for each solution. In this case, the block requires a total of $N * (2^{freqtaps} - 1) * (2^{freqtaps} + 1) + 3$ cycles to complete.

B.2.6 Block 6: PN Sequencer.

Block 6 populates the PRNG which will recreate the sequence used by the transmitter. As stated, Block 5 determined the unique solution. This unique solution

represents the LFSR's value at $t = 0$ or when the first occurrence of the critical frequency was detected. After the first detection, a counter tracks the number of frequency changes the transmitter makes as explained in Section B.2.1. The block 6 clock can operate at one of two frequencies. A fast rate clock allows for the generator to quickly process the sequence until it has executed the same number of times as the transmitter counter has progressed. When the sequences have progressed the same number of operations, the rate changes to that of the transmitter. The output of Block 6 is the complete LFSR value.

B.2.7 Block 7: Frequency Tap.

Block 7 serves as the final process block. The block applies the frequency tap locations determined in Block 5 and applies them to the register value of the newly generated sequence from Block 6. Specifically, each bit located at the determined frequency tap points are concatenated to create the output vector. This vector provides the frequency values to be used by the receiver.

Appendix: VHDL code

C.1 Gold's algorithm code wrapper

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use work.my_package.all;
entity GOLD_wrapper is
  PORT(FREQ_CLK          : in  STD_LOGIC;  -- frequency change rate clk
       FAST_CLK          : in  STD_LOGIC;  -- bus clock
       FFT_IN            : in  STD_LOGIC;
       ASYNC_RST         : in  STD_LOGIC;
       FREQ_OUT          : out STD_LOGIC_VECTOR(4- 1 downto 0);
       Master_Feedback   : in  STD_LOGIC_VECTOR (28-1 downto 0);
       catchup_offset    : in  STD_LOGIC_VECTOR(3 downto 0);
       synced_out        : out STD_LOGIC_VECTOR (0 to 31)
    );
end GOLD_wrapper;

architecture Behavioral of GOLD_wrapper is
----------------------------------------
-- CONSTANTS
----------------------------------------
  constant Master_NumOfFreqTaps  : integer:=4;
  constant Master_Width          : integer:=28;
  constant offset                : integer:=0;
----------------------------------------
-- COMPONENTS
----------------------------------------
  component BUFGMUX
    port(O  : out STD_ULOGIC;
         I0 : in  STD_ULOGIC;
         I1 : in  STD_ULOGIC;
         S  : in  STD_ULOGIC
       );
  end component;

  component matrix_power is
```

```
  generic(Width     : integer := 16 );
  Port(   reset     : in  STD_LOGIC;
          clk       : in  STD_LOGIC;
          feedback  : in  STD_LOGIC_VECTOR (Width-1 downto 0);
          first     : in  STD_LOGIC;
          indexer   : out STD_LOGIC_VECTOR (7 downto 0);
          col_out   : out STD_LOGIC_VECTOR (Width-1 downto 0)
    );
end component matrix_power;

component collect_transpose is
  generic(Width         : integer := 16);
  Port(   freq_clk      : in  STD_LOGIC;
          detect_FFT    : in  STD_LOGIC;
         reset          : in  STD_LOGIC;
         col            : in  STD_LOGIC_VECTOR (Width-1 downto 0);
         indexer        : in  STD_LOGIC_VECTOR (7 downto 0);
         col_out        : out STD_LOGIC_VECTOR (Width-1 downto 0);
         have_first     : out STD_LOGIC;
         matrix_out     : out MATRIX;
         occur_count    : out STD_LOGIC_VECTOR (7 downto 0);
         finished       : out STD_LOGIC;
         occur_indexes  : out STD_LOGIC_VECTOR (127 downto 0)
    );
end component collect_transpose;

component matrix_reduce_3 is
  generic(Width      : integer := 16);
  Port(   matrix_in  : in  MATRIX;
          start      : in  STD_LOGIC;
          reset      : in  STD_LOGIC;
          clk        : in  STD_LOGIC;
          finished   : out STD_LOGIC;
          matrix_out : out MATRIX
    );
end component matrix_reduce_3;

component matrix_solver is
  generic(NumOfFreqTaps : integer := 5;
          Width         : integer := 16
```

```
        );
    port(   reset          : in     STD_LOGIC;
            start          : in     STD_LOGIC;
            clk            : in     STD_LOGIC;
            inputs         : in     MATRIX;
            feedback       : in     STD_LOGIC_VECTOR (Width-1 downto 0);
            finished       : out    STD_LOGIC;
            pulse          : out    STD_LOGIC;
            force_reset    : out    STD_LOGIC;
            output         : buffer STD_LOGIC_VECTOR (Width-1 downto 0);
            taps           : out    STD_LOGIC_VECTOR (Width-1 downto 0)
      );
  end component matrix_solver;

  component pn_gen is
    generic(NumOfFreqTaps: integer := 5;
        Width              : integer := 16);
    port(   clk            : in  STD_LOGIC;
            ShiftEn        : in  STD_LOGIC;
            pulse          : in  STD_LOGIC;
            DataIN_vector  : in  STD_LOGIC_VECTOR (Width-1 downto 0);
            feedback       : in  STD_LOGIC_VECTOR (Width-1 downto 0);
            pn_out         : out STD_LOGIC_VECTOR (Width-1 downto 0)
      );
  end component pn_gen ;

  component freq_tap is
    generic(NumOfFreqTaps : integer := 5;
             Width         : integer := 16);
    port(   PN_seq         : in  STD_LOGIC_VECTOR (Width -1 downto 0);
            taps           : in  STD_LOGIC_VECTOR (Width -1 downto 0);
            freq_out       : out STD_LOGIC_VECTOR (NumOfFreqTaps - 1 downto 0)
      );
  end component freq_tap ;


----------------------------------------
-- INTERCONNECT Signals
----------------------------------------
  signal system_reset               :  STD_LOGIC;
  --power
```

```
  signal power_to_transpose_vector  :  STD_LOGIC_VECTOR (Master_Width-1 downto 0);
  signal power_clk                  :  STD_LOGIC;
  signal power_index                :  STD_LOGIC_VECTOR (7 downto 0);
  --collect
  signal transpose_have_first       :  STD_LOGIC;
  signal transpose_to_reduce_matrix :  MATRIX;
  signal transpose_occur_count      :  STD_LOGIC_VECTOR (7 downto 0);
  signal transpose_col_out          :  STD_LOGIC_VECTOR (Master_Width-1 downto 0);
  signal transpose_finish           :  STD_LOGIC;
  --reduce
  signal reduce_clk                 :  STD_LOGIC;
  signal reduce_to_solver_matrix    :  MATRIX;
  signal reduce_finish              :  STD_LOGIC;
  --solver
  signal solver_to_extender_matrix  :  MATRIX_sol;
  signal solver_finish              :  STD_LOGIC;
  --extender
  signal extender_to_unique_matrix  :  MATRIX_sol_ext;
  signal extender_finish            :  STD_LOGIC;
  --unique
  signal unique_clk                 :  STD_LOGIC;
  signal unique_to_PN_vector        :  STD_LOGIC_VECTOR (Master_Width-1 downto 0);
  signal unique_finish              :  STD_LOGIC;
  signal unique_command_rst         :  STD_LOGIC;
  signal unique_to_pn_pulse         :  STD_LOGIC;
  signal unique_taps_out            :  STD_LOGIC_VECTOR (Master_Width-1 downto 0);
  --pn
  signal pn_clk_sel                 :  STD_LOGIC;
  signal pn_clk                     :  STD_LOGIC;
  signal pn_seq_out                 :  STD_LOGIC_VECTOR (Master_Width-1 downto 0);
  --counter
  signal t0_counter                 :  STD_LOGIC_VECTOR (31 downto 0);
  signal pn_counter                 :  STD_LOGIC_VECTOR (31 downto 0);
  constant zeros                    :  STD_LOGIC_VECTOR (31 downto 0) := (others=> '0');
  signal synced                     :  STD_LOGIC;
  signal reset_counter              :  STD_LOGIC_VECTOR (0 to 7);


BEGIN --architecture
```

```
----------------------------------------
-- instantiations
----------------------------------------
  BUFGMUX_INSTANCE_NAME : BUFGMUX
    port map (  O             => pn_clk_sel,
                I0            => FAST_CLK,
                I1            => FREQ_CLK,
                S             => synced);

  inst_power:  matrix_power
    generic map(Width         => Master_Width)
    Port map(   reset         => system_reset,
                clk           => power_clk,
                feedback      => Master_Feedback,
                first         => transpose_have_first,
                indexer       => power_index,
                col_out       => power_to_transpose_vector
        );

  inst_transpose:  collect_transpose
    generic map(Width         => Master_Width)
    Port map(   freq_clk      => FREQ_CLK,
                detect_FFT    => FFT_IN,
                reset         => system_reset,  --async reset
                col           => power_to_transpose_vector,
                indexer       => power_index,
                have_first    => transpose_have_first,
                matrix_out    => transpose_to_reduce_matrix,
                col_out       => transpose_col_out,
                occur_count   => transpose_occur_count,
                finished      => transpose_finish,
                occur_indexes => OPEN --debug_transpose_index
        );

  inst_reduce:  matrix_reduce_3
    generic map(Width         => Master_Width)
    Port map(  matrix_in      => transpose_to_reduce_matrix,
               start          => transpose_finish,
               reset          => system_reset,
               clk            => reduce_clk,
```

```
            finished        => reduce_finish,
            matrix_out      => reduce_to_solver_matrix
        );

  inst_solver:  matrix_solver
    generic map(NumOfFreqTaps=> Master_NumOfFreqTaps,
                Width         => Master_Width)
    Port map(   reset         => system_reset,
                start         => reduce_finish,
                clk           => FAST_CLK,
                inputs        => reduce_to_solver_matrix,
                finished      => solver_finish,
                output        => unique_to_PN_vector,
                feedback      => Master_Feedback,
                pulse         => unique_to_pn_pulse,
                force_reset   => unique_command_rst,
                taps          => unique_taps_out
        );

  inst_pn_gen:  pn_gen
    generic map(NumOfFreqTaps=> Master_NumOfFreqTaps,
                Width         => Master_Width)
    port map(   clk           => pn_clk_sel,
                ShiftEn       => solver_finish,
                pulse         => unique_to_pn_pulse,
                DataIN_vector => unique_to_PN_vector,
                feedback      => Master_Feedback,
                pn_out        => pn_seq_out
        );

  inst_freq_tap: freq_tap
    generic map(NumOfFreqTaps=> Master_NumOfFreqTaps,
                Width         => Master_Width)
    port map(   PN_seq        => pn_seq_out,
                taps          => unique_taps_out,
                freq_out      => FREQ_OUT
        );

-----------------------------------------
-- Processes
```

```
----------------------------------------
  clks_since_t0proc: process  (power_clk, system_reset) ---counter for matrix multiplier
  begin
    if system_reset = '1' then
      t0_counter <=(others=>'0');
    elsif (rising_edge(power_clk)) then
      if transpose_have_first = '1' then
        t0_counter<=std_logic_vector(unsigned(t0_counter)+1);
      end if;
    end if;
  end process;

  clks_since_ANSproc: process  (pn_clk_sel, system_reset) --counter for PN_seq catchup
  begin
    if system_reset ='1' then
        pn_counter<=(others=>'0');
      elsif (rising_edge(pn_clk_sel)) then
        if (solver_finish='1') then
          pn_counter<=std_logic_vector(unsigned(pn_counter)+1);
        end if;
      end if;
  end process;

  determine_syncproc: process (pn_counter, t0_counter, catchup_offset)  --COMPARITOR
  begin
      if pn_counter = zeros then
        synced<='0';
      elsif (unsigned(t0_counter)<=unsigned(pn_counter)+unsigned(catchup_offset))then
        synced<='1';
      else
        synced<='0';
      --consider adding option
      --if gone to far
      end if;
  end process;

  determine_resetproc: process (unique_command_rst,ASYNC_RST)
  begin
      if ASYNC_RST = '1' then
        reset_counter <= (others => '0');
```

```
      elsif rising_edge(unique_command_rst) then
        reset_counter <= std_logic_vector(unsigned(reset_counter)+1);
      end if;
  end process;

  power_clk    <= FREQ_CLK;
  reduce_clk   <= FAST_CLK;
  unique_clk   <= FAST_CLK;
  system_reset <= ASYNC_RST;

  synced_out(0 to 3)   <= synced & '0' & '0' & '0';
  synced_out(4 to 7)   <= (others=> '0');
  synced_out(8 to 15)  <= reset_counter;
  synced_out(16 to 23) <= transpose_occur_count;
  synced_out(24 to 27) <= '0' & '0' &'0' & '0';
  synced_out(28 to 31) <= solver_finish&reduce_finish&transpose_finish&transpose_have_first;

end Behavioral;
```

C.2 Block 1: Matrix power multiplication

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use ieee.numeric_std.all;
--use ieee.reduce_pkg.all;
--USE IEEE.std_logic_arith.ALL;
--USE IEEE.std_logic_unsigned.ALL;
-- library mylibs_v1_00_a;
-- use mylibs_v1_00_a.my_package.all; --ccm
use work.my_package.all; --ccm

--program to reduce a 16x16 binary matrix to row echelon

entity matrix_power is
  generic(Width : integer := 16 );
    Port (  reset : in  STD_LOGIC;
            clk : in STD_LOGIC;
        feedback: in STD_LOGIC_VECTOR (Width-1 downto 0);
        first: in  STD_LOGIC;
        indexer : out STD_LOGIC_VECTOR (7 downto 0);
```

```
        col_out : out  STD_LOGIC_VECTOR (Width-1 downto 0)
       );
end matrix_power;

architecture Behavioral of matrix_power is
  --type MATRIX is array(15 downto 0) of STD_LOGIC_VECTOR (15 downto 0);
                --
  -- constant s_shift_mat : subMATRIX := (("0000000000000000"), --row 15
                           -- ("1000000000000000"), --row 14
                           -- ("0100000000000000"), --row 13
                           -- ("0010000000000000"), --row 12
                           -- ("0001000000000000"), --row 11
                          -- ("0000100000000000"), --row 10
                          -- ("0000010000000000"), --row 9
                          -- ("0000001000000000"), --row 8
                          -- ("0000000100000000"), --row 7
                          -- ("0000000010000000"), --row 6
                          -- ("0000000001000000"), --row 5
                          -- ("0000000000100000"), --row 4
                          -- ("0000000000010000"), --row 3
                          -- ("0000000000001000"), --row 2
                          -- ("0000000000000100"), --row 1
                          -- ("0000000000000010") ); --row 0

  signal shift_mat : MATRIX := (others => (others => '0'));    --:=(("0000000000000000"), --row 15
                --  ("1000000000000000"), --row 14
                --  ("0100000000000000"), --row 13
                --  ("0010000000000000"), --row 12
                --  ("0001000000000000"), --row 11
                --  ("0000100000000000"), --row 10
                --  ("0000010000000000"), --row 9
                --  ("0000001000000000"), --row 8
                --  ("0000000100000000"), --row 7
                --  ("0000000010000000"), --row 6
                --  ("0000000001000000"), --row 5
                --  ("0000000000100000"), --row 4
                --  ("0000000000010000"), --row 3
                --  ("0000000000001000"), --row 2
                --  ("0000000000000100"), --row 1
                --  ("0000000000000010") ); --row 0
```

```
  signal mat_int : MATRIX:= (others => (others => '0'));     --:=(("0000000000000000"), --row 15
                --  ("1000000000000000"), --row 14
                --  ("0100000000000000"), --row 13
                --  ("0010000000000000"), --row 12
                --  ("0001000000000000"), --row 11
                --  ("0000100000000000"), --row 10
                --  ("0000010000000000"), --row 9
                --  ("0000001000000000"), --row 8
                --  ("0000000100000000"), --row 7
                --  ("0000000010000000"), --row 6
                --  ("0000000001000000"), --row 5
                --  ("0000000000100000"), --row 4
                --  ("0000000000010000"), --row 3
                --  ("0000000000001000"), --row 2
                --  ("0000000000000100"), --row 1
                --  ("0000000000000010") ); --row 0
  signal indexer_i : STD_LOGIC_VECTOR (7 downto 0) := (others=> '0');

begin
  process(clk,reset,feedback) is
    variable temp_row : std_logic_vector(Width-1 downto 0);
    variable temp_row2 : std_logic_vector(Width-1 downto 0);

    variable and_int : std_logic_vector(Width-1 downto 0);
    variable xor_bit : std_logic:='1';

  begin

    if reset = '1' then
      for i in 0 to Width-1 loop -- for each row
        --shift_mat(i)<= s_shift_mat(i) & feedback(i);

        temp_row := (others => '0'); --set all to zero
        temp_row(0) := feedback(i); --set last col to feedback
        if i /= Width - 1 then
          temp_row(i+1) := '1'; --for all but first row, create identity matirx
        end if;
        shift_mat(i)<= temp_row;
```

```
      temp_row2 := (others => '0'); --set all to zero
      temp_row2(i) := '1';
      mat_int(i)  <= temp_row2;

    end loop;
    --col_out <= (others => '0');
    indexer <= (others=>'0');
  elsif rising_edge(clk) then
    if(first = '1') then
      for i in Width-1 downto 0 loop --rows
        for j in Width-1 downto 0 loop --cols
          for h in Width-1 downto 0 loop
            and_int(h):= (mat_int(i)(h) and shift_mat(h)(j));
          end loop;

          xor_bit:='0';
          for k in Width-1 downto 0 loop
            xor_bit:= xor_bit xor and_int(k);
          end loop;
          mat_int(i)(j)<= xor_bit;



          -- mat_int(i)(j) <=((mat_int(i)(15) and shift_mat(15)(j))
                -- xor (mat_int(i)(14) and shift_mat(14)(j))
                -- xor (mat_int(i)(13) and shift_mat(13)(j))
                -- xor (mat_int(i)(12) and shift_mat(12)(j))
                -- xor (mat_int(i)(11) and shift_mat(11)(j))
                -- xor (mat_int(i)(10) and shift_mat(10)(j))
                -- xor (mat_int(i)( 9) and shift_mat( 9)(j))
                -- xor (mat_int(i)( 8) and shift_mat( 8)(j))
                -- xor (mat_int(i)( 7) and shift_mat( 7)(j))
                -- xor (mat_int(i)( 6) and shift_mat( 6)(j))
                -- xor (mat_int(i)( 5) and shift_mat( 5)(j))
                -- xor (mat_int(i)( 4) and shift_mat( 4)(j))
                -- xor (mat_int(i)( 3) and shift_mat( 3)(j))
                -- xor (mat_int(i)( 2) and shift_mat( 2)(j))
                -- xor (mat_int(i)( 1) and shift_mat( 1)(j))
                -- xor (mat_int(i)( 0) and shift_mat( 0)(j)) );
```

```
          end loop;
        end loop;

        indexer <= std_logic_vector(unsigned(indexer_i) + 1);
        indexer_i <= std_logic_vector(unsigned(indexer_i) + 1);
      end if;
    end if;

    --finished <= '1';
  end process;

  get_col: for k in Width-1 downto 0 generate
    col_out(k)<=mat_int(k)(Width-1);
  end generate;
  --col_out<=mat_int(15)(15)& mat_int(14)(15)& mat_int(13)(15)& mat_int(12)(15)& mat_int(11)(15)& mat_int(10)(15)&

end Behavioral;
```

C.3 Block 2: Collection and transposition

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
--USE IEEE.std_logic_arith.ALL;
--USE IEEE.std_logic_unsigned.ALL;
-- library mylibs_v1_00_a;
-- use mylibs_v1_00_a.my_package.all; --ccm
use work.my_package.all; --ccm

--collects 'Width' occurrences, combines to create a matrix and transposes matrix for output

entity collect_transpose is
  generic(Width : integer := 16);
    Port (  freq_clk: in STD_LOGIC;
      detect_FFT : in STD_LOGIC;
      reset: in STD_LOGIC;
      col : in  STD_LOGIC_VECTOR (Width-1 downto 0);
      indexer : in STD_LOGIC_VECTOR (7 downto 0);
      have_first: out STD_LOGIC;
      matrix_out : out MATRIX;
      occur_count : out STD_LOGIC_VECTOR (7 downto 0);
```

```
            finished : out  STD_LOGIC;

      --first_col : out STD_LOGIC_VECTOR (Width-1 downto 0);
      occur_indexes : buffer STD_LOGIC_VECTOR (127 downto 0); --16* 8bit index
      col_out : out  STD_LOGIC_VECTOR (Width-1 downto 0)
       );
end collect_transpose;

architecture Behavioral of collect_transpose is
  --type MATRIX is array(15 downto 0) of STD_LOGIC_VECTOR (15 downto 0);
  signal c_state : integer range 0 to Width+1 := 0; --std_logic_vector(3 downto 0) := "0000";
  --signal occur_indexes_i : STD_LOGIC_VECTOR (127 downto 0);
begin
  process(reset, freq_clk) is
  begin
    if reset='1' then
        have_first <='0';
        finished <='0';
        c_state <= 0;
        matrix_out <= (others => (others => '0'));
        occur_indexes <= (others => '0');

    elsif falling_edge(freq_clk) then
      if detect_FFT = '1' then
        if c_state = 0 then
          --ignore first occurrence
          c_state <= c_state + 1;
          have_first <='1';
          col_out <= col;
          --first_col <= col;

        elsif (c_state < Width+1) then
          matrix_out(c_state-1) <= not(col(Width-1)) & col(Width-2 downto 0);
          c_state <= c_state + 1;
          col_out <= col;
          occur_indexes <= occur_indexes(119 downto 0) & indexer; --will shift all collections into the vecto
          --occur_indexes_i <= occur_indexes_i(119 downto 0) & indexer;
          if(c_state = Width) then
            finished <= '1';
          end if;
```

```
        else
          finished <= '1'; --all have been filled
        end if;
      end if;
    end if;

  end process;


  occur_count <= std_logic_vector(to_unsigned(c_state,8));
end Behavioral;
```

C.4 Block 3: Matrix reduction

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
-- library mylibs_v1_00_a;
-- use mylibs_v1_00_a.my_package.all; --ccm
use work.my_package.all; --ccm

entity matrix_reduce_3 is
  generic(Width  : integer := 16);
    Port (  matrix_in : in MATRIX;
            start : in  STD_LOGIC;
      reset : in STD_LOGIC;
            clk : in STD_LOGIC;
      finished : out std_logic;
      matrix_out : out MATRIX
       );
end matrix_reduce_3;

architecture Behavioral of matrix_reduce_3 is

signal stage: integer range 0 to 6;
signal col_case,row_case,first_one: integer range 0 to width;
signal flag : std_logic;
signal mat_int : MATRIX;
signal temp_row: std_logic_vector(width-1 downto 0);

begin
```

```
process(clk) is
  variable var_vector: std_logic_vector(width-1 downto 0);
  variable var_vector2: std_logic_vector(width-1 downto 0);
  variable var_swap: std_logic_vector(width-1 downto 0);
  begin

  if rising_edge(clk) then
    if reset = '1' then
      --mat_int<=matrix_in;
      col_case<=0;
      stage<=0;
      flag<='0';
      row_case<=width;
      finished<='0';
      first_one <= width-1;

    elsif (start ='1') then                                    --repeat width times: 2n^2+2n
      --stage 1 - find first one, record row number and vector                --width iterations
      --stage 2 - go through all vectors to XOR if it has a one (ignore the selected vector)   --width ite
      --stage 3 - row swap                                        --  1 iteration
      --stage 4 - row elements rotate left                          --  1 iteration
      --stage 5 -
      --stage 6 - hold stage, dont do anything -- requires reset
      if(stage=0) then
        mat_int<=matrix_in; --load initial values
        --col_case<=0;
        stage<=1;
        --flag<='0';
        --row_case<=width;
        --finished<='0';
        col_case<=0;
        flag<='0';
        row_case<=width;
        finished<='0';
        first_one <= width-1;

      elsif (stage=1 or stage = 2) then
        if(row_case>0 and stage = 1) then
          if flag ='0' and row_case> col_case then
            if mat_int(width-1)(width-1) = '1' then
```

```
            temp_row <= mat_int(width-1);
            first_one <= row_case-1;
            flag <='1';  --found a 'one'
          end if;
        end if;
        var_vector:=mat_int(width-1);
        row_case<=row_case-1;
      elsif(row_case>0 and stage = 2) then
        if( (row_case-1 /= first_one) and (mat_int(width-1)(width-1) = '1') ) then
          var_vector:= mat_int(width-1) xor temp_row;
        else
          var_vector:= mat_int(width-1);
        end if;
        row_case<=row_case-1;
      end if;

      if(row_case>0) then
        mat_int(width-1 downto 1) <= mat_int(width-2 downto 0);
        mat_int(0)<= var_vector;
      else  --(row_case=0) then
        if flag= '0' then  --if a one was never found
          temp_row <= (others=> '0');
        end if;
          flag <= '0';
        row_case<=width;
        stage<= stage+1;
        --row_case <= width;
      end if;

    elsif stage = 3 then
      --swap top row and first_one
      if(first_one /= Width-1) then
        var_swap := mat_int(width-1);
        mat_int(width-1) <= mat_int(first_one);
        mat_int(first_one) <= var_swap;
      end if;
      stage<= 4;

    elsif stage = 4 then
      var_vector2:=mat_int(width-1);
```

```
          mat_int(width-1 downto 1) <= mat_int(width-2 downto 0);
          mat_int(0)<= var_vector2;
          first_one <= width-1;
          stage<= 5;

        elsif stage = 5 then
          --rotate left each row by one element
          for i in width-1 downto 0 loop
            mat_int(i)(width-1 downto 1) <= mat_int(i)(width-2 downto 0);
            mat_int(i)(0) <= mat_int(i)(width-1);
          end loop;
          --row_case<= width;
          if col_case < width-1 then
            col_case<= col_case+1;
            stage<= 1;
          else --we have rotated through each element in the rows
            stage <= 6;

          end if;

        elsif stage=6 then
            matrix_out<=mat_int;
            finished <='1';
        end if;
      end if;
    end if;

  end process;
end Behavioral;
```

C.5 Blocks 4 and 5: Solution solver

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
--USE IEEE.std_logic_arith.ALL;
--USE IEEE.std_logic_unsigned.ALL;
-- library mylibs_v1_00_a;
-- use mylibs_v1_00_a.my_package.all; --ccm
use work.my_package.all; --ccm
--use work.std_logic_1164_additions.all;
```

```
--program to reduce a 16x16 binary matrix to row echelon

entity matrix_solver is
  generic(NumOfFreqTaps    : integer := 5;
      Width          : integer := 16
      --Max_Combos     : integer := 31  -- 2^NumOfFreqTaps -1
      );
    Port (  reset : in  STD_LOGIC;
            start : in  STD_LOGIC;
            clk : in STD_LOGIC;
      inputs : in MATRIX;
      --inputs : in  STD_LOGIC_VECTOR (255 downto 0);
      --finished : out STD_LOGIC
      --outputs : out MATRIX_sol --31 sets of 16 bit solution space
      --start : in  STD_LOGIC;--
            --clk : in STD_LOGIC;--
      --inputs : in  MATRIX_sol;--
      feedback: in STD_LOGIC_VECTOR (Width-1 downto 0);
      finished : out STD_LOGIC;--
      pulse : out STD_LOGIC;
      --reset: in STD_LOGIC;--
      force_reset: out STD_LOGIC;
      output : buffer  STD_LOGIC_VECTOR (Width-1 downto 0);
      taps : out     STD_LOGIC_VECTOR (Width-1 downto 0)

       );
end matrix_solver;

architecture Behavioral of matrix_solver is
  --type MATRIX is array(15 downto 0) of STD_LOGIC_VECTOR (15 downto 0);
  --type MATRIX2 is array(31 downto 1) of STD_LOGIC_VECTOR (15 downto 0);  --removed zero vector 31-1
  type FSM_States is (S0,S1,S2,S3,S4a,S4,S5,S6,S7a,S7,S8a,S8,S9);
  signal stage: FSM_States := S0;
  signal row_case: integer range 0 to width := width;
  signal temp_row: STD_LOGIC_VECTOR(Width-1 downto 0) := (others =>'0');
  --signal sol_mask : STD_LOGIC_VECTOR(Width-1 downto 0) := (others =>'0'); --mask to show that these are set
  signal var_loc: STD_LOGIC_VECTOR(Width-1 downto 0) := (others =>'0'); --mash to show these are the variable
  signal var_values: STD_LOGIC_VECTOR(NumOfFreqTaps-1 downto 0) := (others =>'0');
  --type sum_type is array (Width-1 downto 0) of unsigned(4 downto 0);
```

```
    --signal sum : sum_type := (others=> (others=> '0')); --set to max number of 256 --needs enough bits to r
  --signal matrix_in : MATRIX := (others => (others=> '0'));



    signal row_state : integer range 0 to 2**NumOfFreqTaps := 0;
    signal shift_state: integer range 0 to Width := 0;
    signal temp_small : STD_LOGIC_VECTOR (Width-1 downto 0) := (others=>'0');
    signal temp : STD_LOGIC_VECTOR (Width-1 downto 0) := (others=>'0');
    signal occur :STD_LOGIC_VECTOR (2**NumOfFreqTaps downto 1) := (others=>'0');  --had to add an extra front
    signal count: integer range 0 to NumOfFreqTaps := 0;
    --signal stage: integer range 0 to 4;
    signal position:STD_LOGIC_VECTOR (Width-1 downto 0) := (others=>'0');
    signal index: integer range 0 to 2**NumOfFreqTaps-1 := 0;

    signal comp_index: integer range 1 to 2**NumOfFreqTaps-1;
    signal comp_temp: STD_LOGIC_VECTOR (Width-1 downto 0);
    signal comp_ans: STD_LOGIC;
    signal comp_occur: STD_LOGIC;

    constant zeros : STD_LOGIC_VECTOR (2**NumOfFreqTaps-1 downto 1) := (others => '0');
    --constant zeros_width : STD_LOGIC_VECTOR (Width-1 downto 0) := (others => '0');

  constant zeros_tap : STD_LOGIC_VECTOR (NumOfFreqTaps-1 downto 0) := (others => '0');
  constant zeros_width : STD_LOGIC_VECTOR (Width-1 downto 0) := (others => '0');

  signal we: STD_LOGIC :='0';
  signal addr: integer range 0 to 2**NumOfFreqTaps-1 := 2**NumOfFreqTaps-1;
  signal do: STD_LOGIC_VECTOR (Width-1 downto 0) := (others => '0');
  signal di: STD_LOGIC_VECTOR (Width-1 downto 0) := (others => '0');
  constant en: STD_LOGIC :='1';
  signal RAM : MATRIX_sol;
  attribute ram_style: string;
  attribute ram_style of RAM : signal is "block";

begin
  process(clk) is
  --variable tracking : STD_LOGIC_VECTOR(Width-1 downto 0) := (others =>'0'); --shows during loop if the valu
  variable count_n: integer range 0 to Width:=0;
  variable sol_vec: STD_LOGIC_VECTOR(Width-1 downto 0) := (others =>'0');
```

```
variable matrix_out_bit: std_logic :='1';
begin
  if rising_edge(clk) then
    if reset = '1' then
      finished <= '0';
      stage <= S0;
      row_case <= Width;

      force_reset <='0'; --creates the reset pulse through the system
      pulse <='0';
      --finished <='0';
      taps <=(others=>'0');
      output <=(others=>'0');
      --stage <= 0;

    elsif start = '1' then

      case stage is
      when S0 =>
      --if stage = 0 then
        --tracking <= (others =>'0');
        --sol_mask <= (others =>'0');
        var_loc <= (others =>'0');
        matrix_out_bit := '1';
        row_case <= Width;
        stage <= S1;

      when S1 =>
      --elsif stage = 1 then --determines if sum represents a known value or variable
        if(row_case > 0) then
          if(inputs(row_case-1) = zeros_width) then
            var_loc(row_case-1) <= '1';
          end if;
          row_case <= row_case - 1;
        else
          stage <= S2;
          row_case <= Width;
          var_values <= (others => '1');
          --tracking <= sol_mask or var_loc;
        end if;
```

```
when S2 =>
--elsif stage = 2 then

  if (var_values /= zeros_tap) then --loops over 32

    sol_vec := (others=> '0');
    count_n := 0;
    for x in Width-1 downto 0 loop
      if var_loc(x)='1' then
        sol_vec(x) := var_values(NumOfFreqTaps-1-count_n);
        count_n := count_n + 1;
      end if;
    end loop;
    temp_row <= sol_vec;
    stage <= S3;
    row_case <= 0;
    we<='0';
  else
    stage <= S4a;
    we<='0';
    addr<=2**NumOfFreqTaps-1;
  end if;

when S3 =>
--elsif stage = 3 then
  if(row_case < Width ) then
    --loop over i 0:15
    sol_vec := temp_row and inputs(row_case) ;


    matrix_out_bit:='0';
    for k in Width-1 downto 0 loop
      matrix_out_bit:= matrix_out_bit xor sol_vec(k);
    end loop;

    if(inputs(row_case) /= zeros_width) then
      temp_row(row_case)<= matrix_out_bit;
    end if;
```

```
      row_case <= row_case + 1;
    else
      stage <= S2;
      --RAM(to_integer(unsigned(var_values))) <= temp_row; --possibly write to bram
      addr <=to_integer(unsigned(var_values));
      di <=temp_row;
      we<= '1';

      var_values <= std_logic_vector(unsigned(var_values) - 1);
      --row_case <= Width;
    end if;

  when S4a=> stage <= S4;  --cause extra cycle for RAM read
  when S4 =>
  --elsif stage=4+0 then
  --if stage = 0 then
    --step0;
    shift_state <= Width-1;
    row_state <= 2**NumOfFreqTaps-1;
    --load a extended row
    temp_small <= do; --RAM(2**NumOfFreqTaps-1);
    position <= (others=> '0');
    occur <= (others=> '0');
    count <= 0;
    stage <= S5;
    index <= 2**NumOfFreqTaps-1; --31


  when S5 =>
  --elsif stage = 4+1 then
    --step1
    --compare temp to all other short solutions;
    if(index>0) then --repeat through all
      comp_temp <= temp_small;  --reg value
      --comp_index <= index;                --reg input index to compare to
      addr<=index;
      index<=index-1;
    else
      stage <= S6;
      --addr<=row_state;
```

```
    end if;
    occur(index+1) <= comp_ans;--loads value from last compare

  when S6 =>
  --elsif stage = 4+2 then
    --step2
    --did a match occur

    if (comp_occur='1') then     --occur(2**NumOfFreqTaps-1 downto 1) /= zeros) then
      position(shift_state) <= '1';
      count <= count + 1;
    else
      position(shift_state) <= '0';
      count <= count;
    end if;

    -- position(shift_state-1) <= comp_occur;
    -- count <= count + to_integer(comp_occur);

    stage <= S7a;
    addr<=row_state;

  when S7a => stage <= S7;--cause extra cycle for RAM read
  when S7 =>
  --elsif stage = 4+3 then
    --step3  --control section
    if(count = NumOfFreqTaps) then --we have a solution, no more to do
      --done!!!
      output <= do; --RAM(row_state);
      taps <= position;
      pulse <= '1';
      stage <= S9;
    elsif shift_state > 0 then --no solution yet, still have more shifts available
      temp_small <= temp;
      shift_state <= shift_state -1;
      occur <= (others => '0');
      index <= 2**NumOfFreqTaps-1;
      stage <= S5;
    else
      stage<= S8a;
```

```
          if row_state > 1 then
            addr<=row_state-1;
          end if;
        end if;

      when S8a => stage<=S8;  --cause extra cycle for RAM read
      when S8 =>
        if shift_state = 0 and row_state > 1 then  --no more shifts, but other rows
          temp_small <= do;  --RAM(row_state-1);
          shift_state <= Width-1;--reset shift
          row_state<= row_state-1;
          position <= (others=> '0');
          occur <= (others => '0');
          count <= 0;
          index <= 2**NumOfFreqTaps-1;
          stage <= S5;
        else                      --no more rows
          --no solution found
          force_reset<='1';
          stage <= S0;
        end if;

      when S9 =>
      --elsif stage = 4+4 then
        if (output = zeros_width) then
          force_reset<='1';
          stage <= S0;
        else
          --step 4 --
          pulse <= '0';
          finished <='1';
        end if;
      end case;

    end if;
  end if;

end process;

 process (clk)
```

```
  begin
    if clk'event and clk = '1' then
      if en = '1' then
        if we = '1' then
          RAM(addr) <= di;
        end if;
        do <= RAM(addr) ;
      end if;
    end if;
  end process;


  extend: process(temp_small) is
    --variable extended : STD_LOGIC_VECTOR (Width downto (0));--((Width-1) downto (-1*(Width-1))) := (others=
    --variable adding : std_logic_vector(Width-1 downto 0) := (others=>'0');
    variable add_bit: std_logic := '0';
  begin
    --extended := temp_small & '0';
      --feedback taps on 3,5,7,11,13
      --for j in Width-2 downto 0 loop
      --adding := feedback and temp_small;
        add_bit:= '0';
        for x in Width-1 downto 0 loop
          add_bit:= add_bit xor (feedback(x) and temp_small(x));
        end loop;
        --extended(0) := add_bit;
      --end loop;
    temp <= temp_small(Width-2 downto 0) & add_bit;
  end process;

  comp_ans <= '1' when (comp_temp = do) else '0'; --RAM(comp_index)) else '0';
  comp_occur<= '0' when occur(2**NumOfFreqTaps-1 downto 1) = zeros else '1';


end Behavioral;
```

C.6 Block 6: PN Sequencer

```
--The following is example code that implements two LFSRs which can be used as part of pn generators.
--The number of taps, tap points, and LFSR width are parameratizable. When targetting Xilinx (Virtex)
--all the latest synthesis vendors (Leonardo, Synplicity, and FPGA Express) will infer the shift
```

```
--register LUTS (SRL16) resulting in a very efficient implementation.
--
--Control signals have been provided to allow external circuitry to control such things as filling,
--puncturing, stalling (augmentation), etc.
--
--Mike Gulotta
--11/4/99
--Revised 3/17/00:  Fixed "commented" block diagram to match polynomial.
--Adapted 7/10/2013 by Capt Curtis Medve

library ieee ;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pn_gen is
  generic(NumOfFreqTaps    : INTEGER := 5;   -- # of taps for I channel LFSR, including output tap.
       Width          : INTEGER := 16); -- LFSR length (ie, total # of storage elements)
  port( clk        : in     STD_LOGIC;
       ShiftEn     : in     STD_LOGIC;
       pulse      : in  STD_LOGIC;
       DataIN_vector    : in  STD_LOGIC_VECTOR (Width-1 downto 0);
       feedback     : in  STD_LOGIC_VECTOR (Width-1 downto 0);
       pn_out              : out STD_LOGIC_VECTOR(Width-1 downto 0)
      );
end pn_gen ;


architecture rtl of pn_gen is

  type TapPointArray_i is array (NumOfFreqTaps-1 downto 0) of integer;

  -- Parameratize I LFSR taps.  (e.g.  I(x) = X**17 + X**5 + 1)
  -- Plug in I channel tap points, including output tap 0.
  --constant Tap_i : TapPointArray_i := (13,11,9,5,3);  --relative it 15:0  or reverse of 3,5,7,11,13

  signal srl_i          : STD_LOGIC_VECTOR(Width-1 downto 0);       -- shift register.
  signal par_fdbk_i     : STD_LOGIC_VECTOR(Width-1+1 downto 0);   -- Parity feedback.
  signal lfsr_in_i      : STD_LOGIC;           -- mux output.
  --signal start_flag    : STD_LOGIC;
```

```
begin

-------------------------------------------------------------------
------------------ I Channel ------------------------------------
-------------------------------------------------------------------

  Shift_i : process (clk,pulse,DataIN_vector)
  begin
  if pulse='1' then
    srl_i <= DataIN_vector;

  elsif rising_edge(clk) then
      if ShiftEn = '1' then
        srl_i <= srl_i(Width-1-1 downto 0)& lfsr_in_i;
      end if;
  end if;
  end process;

  par_fdbk_i(0) <= '0';
  fdbk_i : for X in 0 to Width-1 generate -- parity generator
      par_fdbk_i(X+1) <= par_fdbk_i(X) xor (srl_i(X) and feedback(X));
  end generate fdbk_i;

  --lfsr_in_i <= DataIn_i when FillSel = '1' else par_fdbk_i(par_fdbk_i'high);
  lfsr_in_i <= par_fdbk_i(Width);

  pn_out <= srl_i;  -- PN I channel output.
-------------------------------------------------------------------

end rtl;
```

C.7 Block 7: Frequency Tap

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity freq_tap is
  generic(NumOfFreqTaps : integer := 5;   -- # of freq taps
          Width         : integer := 16); -- length of input vector
  port(   PN_seq        : in std_logic_vector (Width -1 downto 0);
```

```
          taps          : in std_logic_vector (Width -1 downto 0);
          freq_out      : out std_logic_vector(NumOfFreqTaps - 1 downto 0)
     );
end freq_tap ;

architecture behavioral of freq_tap is
begin
  get_them : process (PN_seq, taps)
    variable temp_output : std_logic_vector(NumOfFreqTaps - 2 downto 0) := (others=>'0');
  begin
    for i in Width-2 downto 0 loop
      if taps(i)='1' then
        temp_output := temp_output(NumOfFreqTaps - 3 downto 0) & PN_seq(i);
      end if;
    end loop;
    freq_out<= PN_seq(Width-1) & temp_output;
end process;
end behavioral;
```

References

[1] "WARP Project". URL http://warpproject.org.

[2] Abdel Rahman, Mohammad J, Hanif Rahbari, and Marwan Krunz. "Adaptive frequency hopping algorithms for multicast rendezvous in DSA networks". *Dynamic Spectrum Access Networks (DYSPAN), 2012 IEEE International Symposium on*, 517–528. IEEE, 2012.

[3] Air Force Chief Scientist, Office of the. *Report on Technology Horizons: A Vision for Air Force Science & Technology During 2010–2030*, 52–55, 60, 78–91. 2010.

[4] Bian, Kaigui and Jung-Min Park. "Asynchronous channel hopping for establishing rendezvous in cognitive radio networks". *INFOCOM, 2011 Proceedings IEEE*, 236–240. 2011. ISSN 0743-166X.

[5] Bian, Kaigui, Jung-Min Park, and Ruiliang Chen. "A quorum-based framework for establishing control channels in dynamic spectrum access networks". *Proceedings of the 15th annual international conference on Mobile computing and networking*, 25–36. ACM, 2009.

[6] Bose, Vanu, David Wetherall, and John Guttag. "Next century challenges: RadioActive networks". *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, 242–248. ACM, 1999.

[7] DaSilva, L.A. and I. Guerreiro. "Sequence-Based Rendezvous for Dynamic Spectrum Access". *New Frontiers in Dynamic Spectrum Access Networks, 2008. DySPAN 2008. 3rd IEEE Symposium on*, 1–7. 2008.

[8] De Domenico, Antonio, Emilio Calvanese Strinati, and M-G Di Benedetto. "A survey on MAC strategies for cognitive radio networks". *Communications Surveys & Tutorials, IEEE*, 14(1):21–44, 2012.

[9] Faint, Stephanie, Oktay Ureten, and Tricia Willink. "Impact of the number of sensors on the network cost and accuracy of the radio environment map". *Electrical and Computer Engineering (CCECE), 2010 23rd Canadian Conference on*, 1–5. IEEE, 2010.

[10] Federal Communications Commission. *Notice of Inquiry: Promoting More Efficient Use of Spectrum THrough Dynamic Spectrum Use Technologies, FCC 10-198*, Nov 2010.

[11] Federal Communications Commission. "FCC Encyclopedia", December 2013. URL http://www.fcc.gov/encyclopedia/accessing-spectrum.

[12] Gast, Matthew S. *802.11 Wireless Networks: The Definitive Guide, Second Edition*. O'Reilly Media, Inc., 2005. ISBN 0596100523.

[13] Gold, Robert. "Method and system for synchronizing and selectively addressing multiple receivers in a wireless, spread spectrum communication system", 06 2008.

[14] Hanif, Muhammad Fainan, Peter J Smith, and Mansoor Shafi. "Performance of cognitive radio systems with imperfect radio environment map information". *Communications Theory Workshop, 2009. AusCTW 2009. Australian*, 61–66. IEEE, 2009.

[15] Htike, Zaw and Choong Seon Hong. "Neighbor discovery for cognitive radio ad hoc networks". *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '13, 102:1–102:6. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-1958-4. URL http://doi.acm.org/10.1145/2448556.2448658.

[16] Lin, Zhiyong, Hai Liu, Xiaowen Chu, and Yiu-Wing Leung. "Jump-stay based channel-hopping algorithm with guaranteed rendezvous for cognitive radio networks". *INFOCOM, 2011 Proceedings IEEE*, 2444–2452. IEEE, 2011.

[17] Liu, Hai, Zhiyong Lin, Xiaowen Chu, and Y.-W. Leung. "Ring-Walk Based Channel-Hopping Algorithms with Guaranteed Rendezvous for Cognitive Radio Networks". *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, 755–760. 2010.

[18] Liu, Hai, Zhiyong Lin, Xiaowen Chu, and Yui-Wing Leung. "Taxonomy and challenges of rendezvous algorithms in cognitive radio networks". *Computing, Networking and Communications (ICNC), 2012 International Conference on*, 645–649. 2012.

[19] Marinho, José and Edmundo Monteiro. "Cognitive radio: survey on communication protocols, spectrum decision issues, and future research directions". *Wireless Networks*, 18(2):147–164, 2012.

[20] McHenry, Mark A. "NSF spectrum occupancy measurements project summary". *Shared spectrum company report*, August 2005.

[21] McHenry, Mark A, Dan McCloskey, Dennis Roberson, and John T. MacDonald. "Spectrum Occupancy Measurements: Chicago, Illinois". *Shared spectrum company report*, December 2005.

[22] Mclean, Ryan K., Mark D. Silvius, Kenneth M. Hopkinson, Bridget N. Flatley, Ethan S. Hennessey, Curtis C. Medve, Jared J. Thompson, Matthew R. Tolson, and Clark V. Dalton. "An Architecture for Coexistence with Multiple Users in Frequency Hopping Cognitive Radio Networks". *Selected Areas in Communications, IEEE Journal on*, 32(3):To appear, 2014. ISSN To appear.

[23] Mitola, J. and Jr. Maguire, G.Q. "Cognitive radio: making software radios more personal". *Personal Communications, IEEE*, 6(4):13–18, 1999. ISSN 1070-9916.

[24] Mitola, Joseph III. *Cognitive radio: An Integrated Agent Architecture for Software Defined Radio*. Ph.D. thesis, Royal Institute of Technology, May 2000.

[25] National Telecommunications and Information Administration. *Manual of Regulations and Procedures for Federal Radio Frequency Management*, May 2013.

[26] Peterson, Roger L., Rodger E. Ziemer, and David E. Borth. *Introduction to Spread Spectrum Communications*. Prentice Hall, 1995.

[27] Proakis, John. *Digital communications*. McGraw-Hill, Boston, 2008. ISBN 978-0-07-295716-7.

[28] Robert Gold Comm Systems Inc. *Frequency Hopping Signal Prediction and Countermeasures*. Technical Report AFRL-SN-WP-TR-1999-1053, Air Force Research Laboratory, December 1998.

[29] Robert Gold Comm Systems Inc. *Modern Network Command and Control Warfare*. Technical Report AFRL-SN-WP-TR-2003-1148, Air Force Research Laboratory, August 2003.

[30] Robertson, Andrew, Lan Tran, Joseph Molnar, and Er-Hsien Frank Fu. "Experimental comparison of blind rendezvous algorithms for tactical networks". *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, 1–6. IEEE, 2012.

[31] Secretary of the Air Force, Office of the. *Fiscal Year 2010 Air Force Posture Statement*, Feb 2010.

[32] Shared Spectrum Company. "General Survey of Radio Frequency Bands (30 MHz to 3 GHz): Vienna, Virginia". *Shared spectrum company report*, September 2010.

[33] Shin, Jongmin, Dongmin Yang, and Cheeha Kim. "A Channel Rendezvous Scheme for Cognitive Radio Networks". *Communications Letters, IEEE*, 14(10):954–956, 2010. ISSN 1089-7798.

[34] Shokri-Ghadikolaei, Hossein and Rajab Fallahi. "Intelligent Sensing Matrix Setting in Cognitive Radio Networks". 2012.

[35] Theis, N.C., R.W. Thomas, and L.A. DaSilva. "Rendezvous for Cognitive Radios". *Mobile Computing, IEEE Transactions on*, 10(2):216–227, 2011. ISSN 1536-1233.

[36] Wang, Beibei and KJ Ray Liu. "Advances in cognitive radio networks: A survey". *Selected Topics in Signal Processing, IEEE Journal of*, 5(1):5–23, 2011.

[37] Wellens, Matthias and Petri Mähönen. “Lessons learned from an extensive spectrum occupancy measurement campaign and a stochastic duty cycle model”. *Mobile networks and applications*, 15(3):461–474, 2010.

[38] Williams, John R. “U.S. Spectum Allocations 300-3000 MHz”. *Federal Communications Commission report*, November 2002.

[39] Wu, Ching-Chan and Shan-Hung Wu. “On bridging the gap between homogeneous and heterogeneous rendezvous schemes for cognitive radios”. *Proceedings of the fourteenth ACM international symposium on Mobile ad hoc networking and computing*, 207–216. ACM, 2013.

[40] Xilinx, Inc. *ML507 Evaluation Platform User Guide*. UG347. 2011.

[41] Xin, C., M. Song, L. Ma, and C.-C. Shen. “ROP: Near-Optimal Rendezvous for Dynamic Spectrum Access Networks”. *Vehicular Technology, IEEE Transactions on*, 62(7):3383–3391, 2013. ISSN 0018-9545.

[42] Zhang, Yifan, Gexin Yu, Qun Li, Haodong Wang, Xiaojun Zhu, and Baosheng Wang. “Channel-Hopping-Based Communication Rendezvous in Cognitive Radio Networks”. 2013.

[43] Zhao, Youping, Bin Le, and Jeffrey H Reed. “Network support–The radio environment map”. *Cognitive radio technology*, 325–366, 2006.

[44] Zhao, Youping, Jeffrey H Reed, Shiwen Mao, and Kyung K Bae. “Overhead analysis for radio environment map enabled cognitive radio networks”. *Networking Technologies for Software Defined Radio Networks, 2006. SDR’06.1 st IEEE Workshop on*, 18–25. IEEE, 2006.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
27–03–2014	Master's Thesis	Oct 2013–Mar 2014

4. TITLE AND SUBTITLE

Estimation and Coordination of Sequence Patterns for Frequency Hopping Dynamic Spectrum Access Networks

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Medve, Curtis C., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-14-M-52

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Vasu Chakravarthy
2241 Avionics Circle
WPAFB, OH 45433
vasu.chakravarthy@us.af.mil
937-528-8269

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RYWE

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In 2010, the Shared Spectrum Company showed in a survey of Radio Frequency (RF) bands that underutilization of spectrum has resulted from current frequency management practices. Traditional frequency allocation allows large bands of licensed spectrum to remain vacant even under current high demands. Cognitive radio's (CR) use of Dynamic Spectrum Access (DSA) enables better spectrum management by allowing usage in times of spectrum inactivity. This research presents the CR problem of rendezvous for fast Frequency Hopping Spread Spectrum (FHSS) networks, and examines protocols for disseminating RF environment information to coordinate spectrum usage. First, Gold's algorithm is investigated as a rendezvous protocol for networks utilizing fast frequency hopping. A hardware implementation of Gold's algorithm on a Virtex-5 Field Programmable Gate Array (FPGA) is constructed to determine the resource requirements and timing limitations for use in a CR. The resulting design proves functionality of the algorithm, and demonstrates a decrease in time-to-rendezvous over current methods. Once a CR network is formed, it must understand the changing environment in order to better utilize the available spectrum. This research addresses the costs a network incurs to coordinate such environment data. Three exchange protocols are introduced and evaluated via simulation to determine the best technique based on network size. The resulting comparison found that smaller networks function best with polled or time-division based protocols where radios always share their environment information. Larger networks, on the other hand, function best when a dispute-based exchange protocol was utilized. These studies together conclude that the selection of a rendezvous algorithm or a protocol for the exchange of environment data in a CR network are determined by the characteristics of the network, and therefore their selection requires a cognitive decision.

15. SUBJECT TERMS

Cognitive Radio, Frequency Hopping, Sequence Estimation, Dynamic Spectrum Access, Rendezvous

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			LTC Robert J. McTasney, AFIT/ENG
U	U	U	UU	201	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x4460 robert.mctasney@afit.edu